VMIVME-2540
Intelligent Counter/Controller

Product Manual

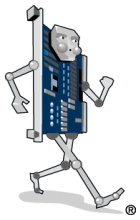
A GE Fanuc Company

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(256) 880-0444 ◆ (800) 322-3616 ◆ Fax: (256) 882-0859

500-002540-000 Rev. N

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(256) 880-0444 ◆ (800) 322-3616 ◆ Fax: (256) 882-0859

COPYRIGHT AND TRADEMARKS



(I/O man figure)

(IOWorks man figure)

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(256) 880-0444 ◆ (800) 322-3616 ◆ Fax: (256) 882-0859

Table of Contents

List of Figures 9

List of Tables 11

Overview 13
Disclaimer: Notice About Equivalent Parts 14
Reference Material List 15
Physical Description and Specifications 15
Safety Summary 16
Ground the System 16
Do Not Operate in an Explosive Atmosphere 16
Keep Away from Live Circuits 16
Do Not Service or Adjust Alone 16
Do Not Substitute Parts or Modify System 16
Dangerous Procedure Warnings 16
Safety Symbols Used in This Manual 17

Chapter 1 - Theory of Operation 19
Functional Description 19
System Timing Controller Front-End Logic 19
RS-422 Line Driver and Receiver 19
Synchronizer and Conditioning Logic 20
AM9513A System Timing Controller 22
QPM Direction Change Interrupt Logic 23
I/O Processor 24
68HC000 CPU 24
Decode and Control Logic 24
Local Bus Arbitration 24

Local Address Decode 26
Local I/O Functions. 28
Local Memory 30
EPROM Firmware. 30
Static RAM 30
Local Interrupt Controller. 31
STC Interrupts. 31
VMEbus Command Interrupt. 31
VMEbus Slave Interface 33
VMEbus Slave Address Decode. 33
Command Status Code. 33
VMEbus Interrupter Modules. 35

Chapter 2 - Configuration and Installation 37
Unpacking Procedures 38
Configuration. 39
DIP Switch Settings. 40
Jumper Options. 40
I/O Connector Pin Assignments 42
Recommended Discrete Wire Connectors and Terminal Blocks. 45
TTL/Single-Ended Input Signal Compatibility Configuration 46

Chapter 3 - Programming 49
VMEbus Interface Memory Map 51
Board ID/Configuration Buffer. 52
Firmware Revision Level. 53
Command Code 53
Command Status Code. 53
Command Status Interrupt Request Level 54
Command Status Interrupt Vector. 55
Channel ID 55
Continuous/Discrete Flag 55
Measurement Ready Flag. 56
Measurement Channel ID. 56
Channel Measurement Status. 56
Channel Control Block Registers 57
Timer Channel Control Block 57
VMIVME-2540 Continuous/Discrete Flag Buffer. 58
VMIVME-2540 Measurement Data Valid Flags Buffer 58

VMIVME-2540 Firmware Release Information. 58
VMIVME-2540 Daignostic Buffer 59
Command Interface 60
Programming Using the Command Interface. 61
Command Status Codes 63
Modes of Operation 64
Input Modes of Operation. 65
Output Modes of Operation 66
Timing Modes of Operation 66
Channel Control Blocks Common Parameters 67
Operation Mode Selection Flag 68
. . . . . Format of the Operation Mode Select Flag:69
Operational Mode Select Flag 69
Command Descriptions 70
Initialization and Synchronization Command Codes 70
Channel Input/Measurement Command Codes. 71
Integer 16-bit Event Counting 71
Integer 32-bit Event Counting 72
Period Measurement 74
Frequency Measurement. 75
Pulse-Width Measurement 76
Quadrature Position Measurement 78
Integer Quadrature Position Measurement 81
16-bit Integer Period Measurement 82
32-bit Integer Period Measurement 83
32-bit Integer Pulse-Width Measurement 85
Group Acquisition Mode (Integer QPM). 86
16-bit Integer Pulse Measurement 87
Delayed Event Timer with VMEbus Interrupt. 88
Programming Strategies for Input Operations 89
Continuous Data Acquisition Mode 90
Discrete Data Acquisition Mode 91
Channel Output/Waveform Generation Command Codes. 93
16-bit Frequency Divider 93
32-bit Frequency Divider 94
Period/Pulse-Width Generation 95
Frequency/Duty Cycle Generation 96
Pulse Sequence Generation 96

Programmed Output Mode 97
Quadrature Position Control 98
Programming Strategies for Output Operations 99
Timer Operation Command Codes 99
Timer/Periodic Interrupt 99
Auxiliary Commands 100
Getting Started 102

Appendix A - Example Code 117
Terminal Output of Program gs.c 118
Programming Example 120

List of Figures

Figure 1-1 VMIVME-2540 Intelligent Counter/Controller 21
Figure 1-2 VMIVME-2540 System Timing Controller 22
Figure 1-3 QPM Direction Change Interrupt Logic 23
Figure 1-4 Local Bus Arbitration 25
Figure 1-5 Interrupt Controller for STC Outputs 32
Figure 1-6 VMEbus Slave Address Decode 34
Figure 1-7 Intelligent Interface Interrupter Module 35
Figure 2-1 VMIVME-2540 DIP Switches and Jumper Options 39
Figure 2-2 Example DIP Switch Settings 40
Figure 2-3 P3/P4 Pinout Layout 44
Figure 2-4 Typical RS-422-Compatible Signal Connections 47
Figure 2-5 Typical TTL-Compatible Signal Connections 47

List of Tables

Table 1-1 VMIVME-2540 Local CPU Address Map 26
Table 1-2 VMIVME-2540 Local I/O Address Map 27
Table 2-1 RS-422 Receiver Termination Jumper Options 40
Table 2-2 Configuration Jumpers E50 and E51 41
Table 2-3 VMEbus Access Select Jumpers E52, E53 and E54 41
Table 2-4 I/O Connector P3 Pin Assignments 42
Table 2-5 I/O Connector P4 Pin Assignments 43
Table 2-6 Recommended Discrete Wire Connectors and Accessories 45
Table 2-7 TTL/Single-Ended Input Signal Compatibility Configuration 46
Table 3-1 VMIVME-2540 VMEbus I/F Addresses 51
Table 3-2 Board ID/Configuration (Offset $0000) 52
Table 3-3 VMIVME-2540 ID/Configuration Values 52
Table 3-4 Firmware Revision Level (Offset $0002) 53
Table 3-5 Command Code (Offset $0004) 53
Table 3-6 Command Status Code (Offset $0006) 54
Table 3-7 Command Status Interrupt Request Level (Offset $0008) 54
Table 3-8 Command Status Interrupt Vector (Offset $0009) 55
Table 3-9 Channel ID (Offset $000A) 55
Table 3-10 Continuous/Discrete Flag (Offset $000B) 55
Table 3-11 Measurement Ready Flag (Offset $000C) 56
Table 3-12 Measurement Channel ID (Offset $000E) 56
Table 3-13 Channel Measurement Status (Offset $000F) 56
Table 3-14 Channel Control Block Registers 57
Table 3-15 Timer Channel Control Block Registers 57
Table 3-16 VMIVME-2540 Continuous/Discrete Flag Buffer 58
Table 3-17 VMIVME-2540 Measurement Data Valid Flags Buffer 58
Table 3-18 VMIVME-2540 Firmware Release Information 58

Table 3-19 VMIVME-2540 Diagnostic Buffer . . . 59
Table 3-20 VMIVME-2540 Host Commands . . . 60
Table 3-21 VMIVME-2540 Status Codes . . . 63
Table 3-22 Front Panel External Clock, Gate, and Output Connections . . . 64
Table 3-23 Typical CCB Format/Common Parameters . . . 67
Table 3-24 Gate/Edge Codes . . . 68
Table 3-25 Clock Period (Time Base) Select Code . . . 68
Table 3-26 Operational Mode Select Flag . . . 69
Table 3-27 16-bit Event Counter Channel Control Block . . . 71
Table 3-28 32-bit Event Counter Channel Control Block . . . 73
Table 3-29 Period Measurement Channel Control Block . . . 75
Table 3-30 Frequency Measurement Channel Control Block . . . 76
Table 3-31 Pulse-Width Measurement Channel Control Block . . . 77
Table 3-32 QPM Channel Control Block . . . 79
Table 3-33 QPM Channel Status Codes . . . 79
Table 3-34 Integer QPM Channel Control Block . . . 81
Table 3-35 Integer Period Measurement Channel Control Block . . . 83
Table 3-36 32-bit Integer Period Measurement Channel Control Block . . . 84
Table 3-37 32-bit Pulse-Width Measurement Channel Control Block . . . 86
Table 3-38 16-bit Integer Pulse-Width Measurement Channel Control Block . . . 87
Table 3-39 Delayed Event Timer CCB Format . . . 89
Table 3-40 16-bit Frequency Divider Channel Control Block . . . 94
Table 3-41 32-bit Frequency Divider Channel Control Block . . . 94
Table 3-42 Period/Pulse-Width Generation Channel Control Block . . . 95
Table 3-43 Frequency/Duty Cycle Generation Channel Control Block . . . 96
Table 3-44 Pulse Sequence CCB Parameters . . . 97
Table 3-45 Programmed Output CCB Parameters . . . 97
Table 3-46 Quadrature Control Output Mode . . . 98
Table 3-47 Timer/Periodic Interrupt Channel Control Block . . . 100
Table 3-48 Block Move Diagnostic Buffer Entries . . . 101
Table 3-49 Diagnostic Buffer Entry for Execute Command . . . 101
Table 3-50 VMIVME-2540 Configuration Using Example Set 1 . . . 102
Table 3-51 VMIVME-2540 Configuration Using Example Set 2 . . . 102
Table 3-52 Wire Connections - Example Set 1 . . . 103
Table 3-53 Wire Connections - Example Set 2 . . . 103

Overview

Contents

Reference Material List 15
Safety Summary 16
Safety Symbols Used in This Manual 17

Introduction

The VMIVME-2540 is an intelligent digital input / output board with a VMEbus slave interface which optionally provides 4, 8, 16, or 24 channels of digital signal measurement and signal generation capability. These measurement and signal output functions are implemented by an array of AM9513A system timing controllers (STC) and associated interface logic. An on-board VMIVME-2540 local CPU (68HC000 Microprocessor) relieves the user from the task of programming the system timing controllers directly by providing a command-driven user interface to perform the board functions. The user configures measurement, timing and output functions for each channel through a simple, memory-mapped interface consisting of commands, command status, parameters, and return measurements.

Each channel of the VMIVME-2540 consists of a 16-bit counter with clock and gate inputs and a digital waveform output. The modes of operation for the unit are:

- Event counter with programmable limit count and optional gate
- Frequency divider
- Event-triggered interrupt with delay
- Timer / periodic VMEbus interrupt
- Period / frequency / pulse-width measurement
- Square wave / pulse train generation
- Quadrature position measurement
- Quadrature position control

Channels may be grouped in pairs for 32-bit extended precision and extended measurement modes. The channels are all buffered at the VMIVME-2540 front panel

with RS-422 line receivers and drivers. Support for single-ended TTL inputs is provided by an on-board TTL threshold voltage which may be connected externally to the inverting input of the RS-422 line receiver.

The VMEbus interface of the VMIVME-2540 consists of a 64 Kbyte static RAM memory, a command status buffer, and two interrupter modules. The 64 Kbyte memory, shared between the VMEbus and the VMIVME-2540 local CPU, is used to exchange command, command status, and parametric and measurement data. The command status buffer overlays a one word memory location with no local bus arbitration for VMEbus access to the status data. VMEbus access to the command status buffer occurs with minimal delay. The interrupter modules on the VMIVME-2540 may assert up to two VMEbus interrupts simultaneously on interrupt levels IRQ1 through IRQ7 with independent 8-bit vectors.

Disclaimer: Notice About Equivalent Parts

"In the sections which follow, reference to specific part types do not guarantee the usage of that part in the final assembly. Equivalent Integrated Circuits may be substituted in the final assembly without notification."

Reference Material List

Refer to *The VMEbus Specification* for a detailed explanation of the VMEbus. *The VMEbus Specification* is available from the following source:

VITA
VMEbus International Trade Association
7825 East Gelding Dr. Suite 104
Scottsdale, AZ 85260-3415
(602) 951-8866
FAX: (602) 951-0720
Email: info@vita.com

Physical Description and Specifications

Refer to 800-002540-000 specification available from VMIC.

VMIC
12090 South Memorial Pkwy.
Huntsville, AL 35803-3308, USA
(256) 880-0444
(800) 322-3616
FAX: (256) 882-0859
www.vmic.com

Safety Summary

The following general safety precautions must be observed during all phases of the operation, service, and repair 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 this product.

VMIC assumes no liability for the customer's failure to comply with these requirements.

Ground the System

To minimize shock hazard, the chassis and system cabinet must be connected to an electrical ground. A three-conductor AC power cable should be used. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet.

Do Not Operate in an Explosive Atmosphere

Do not operate the system in the presence of flammable gases or fumes. Operation of any electrical system in such an environment constitutes a definite safety hazard.

Keep Away from Live Circuits

Operating personnel must not remove product covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Do Not Service or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts or Modify System

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to VMIC for service and repair to ensure that safety features are maintained.

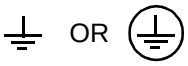
Dangerous Procedure Warnings

Warnings, such as the example below, precede only potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

STOP: Dangerous voltages, capable of causing death, are present in this system. Use extreme caution when handling, testing, and adjusting.

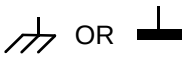
Safety Symbols Used in This Manual

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1,000 V are so marked).



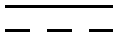
Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating equipment.

Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. Before operating the equipment, terminal marked with this symbol must be connected to ground in the manner described in the installation (operation) manual.



Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.

Alternating current (power line).



Direct current (power line).

Alternating or direct current (power line).

STOP: Informs the operator that a practice or procedure should not be performed. Actions could result in injury or death to personnel or could result in damage to or destruction of part or all of the system.

WARNING: Denotes a hazard. It calls attention to a procedure, a practice, a condition, which, if not correctly performed or adhered to, could result in injury or death to personnel.

CAUTION: Denotes a hazard. It calls attention to an operating procedure, a practice, or a condition, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the system.

NOTE: Denotes important information. It calls attention to a procedure, a practice, a condition or the like, which is essential to highlight.

CHAPTER

1

Theory of Operation

Contents

I/O Processor 24
Local Memory 30
Local Interrupt Controller 31
VMEbus Slave Interface 33
VMEbus Interrupter Modules 35

Introduction

Functional Description

A block diagram of the VMIVME-2540 Intelligent Counter/Controller is shown in Figure 1-1 on page 21. The functions of the board may be divided into five groups:

- System timing controller front-end logic
- VMIVME-2540 local CPU interrupt controller
- VMIVME-2540 local CPU/memory/interface support logic
- VMEbus Slave DTB Interface
- VMEbus Interrupter Modules

Each of these groups will be discussed in more detail in the following sections.

System Timing Controller Front-End Logic

The system timing controller front-end logic consists of RS-422 buffers, synchronizing and conditioning logic, quadrature position measurement logic (QPM), and the AM9513A system timing controller, as shown in Figure 1-2 on page 22.

RS-422 Line Driver and Receiver

The RS-422 line drivers used on the VMIVME-2540 have a specified current sink/source value of ±60 mA and 2.0 V minimum differential output voltage. The RS-422 line receivers have a differential input voltage ±25 V maximum. Each input is

optionally terminated with a 120 Ohm resistor (See “Jumper Options” on page 40). TTL input signal compatibility is supported by disconnecting the termination resistor, connecting TTL compatibility voltage (VTTL) at the P3 and P4 I/O connectors to the inverting input of the RS-422 line receiver, and driving the noninverting input of the RS-422 line receiver with the TTL signal. The on-board threshold voltage source is a regulated source with output current 100 mA maximum and output voltage 1.4 V nominal. Refer to “Jumper Options” on page 40 and “TTL/Single-Ended Input Signal Compatibility Configuration” on page 46 for more information about TTL applications.

Synchronizer and Conditioning Logic

Each group of four clock inputs to the VMIVME-2540 are synchronized to the 5 MHz STC clock by octal registers. The synchronized clock signals feed into the source-gate conditioning logic, implemented in a programmable logic device (PLD). Each group of four gate inputs to the VMIVME-2540 are data inputs to D-flip-flop (DFF) macrocells in the source-gate conditioning logic with the DFF macrocells clocked by the corresponding synchronous clock signal. Therefore, the gate inputs to the AM9513A are guaranteed to meet setup and hold requirements relative to the corresponding synchronous clock signal and the 5 MHz STC clock.

This conditioning logic is mode dependent. There is a quadrature position measurement (QPM) mode register bit which corresponds to each pair of adjacent even/odd pair channels. When a mode register bit is zero (nonquadrature position measurement mode), the two synchronous source signals pass on directly to the AM9513A system timing controller. When a mode register bit is one, the two synchronized clock sources pass through edge-detection logic which creates a 100 ns pulse for every source edge. The quadrature clock inputs are analyzed for direction, and this direction signal selects which input to the AM9513A STC receives an edge pulse: for 16-bit quadrature position measurement, even channel numbers may be considered *clockwise*, and odd channel numbers considered *counterclockwise*.

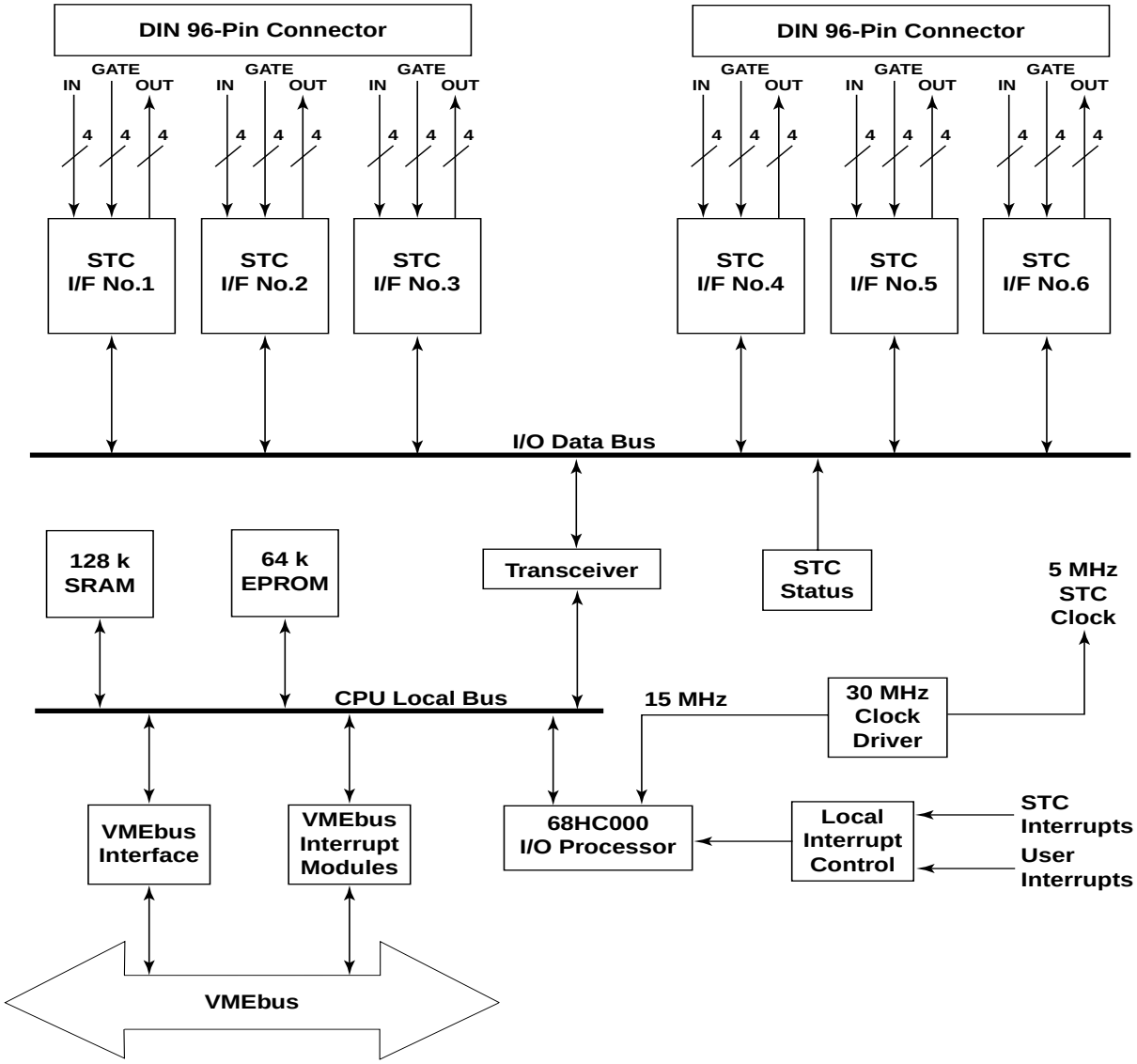


Figure 1-1 VMIVME-2540 Intelligent Counter/Controller

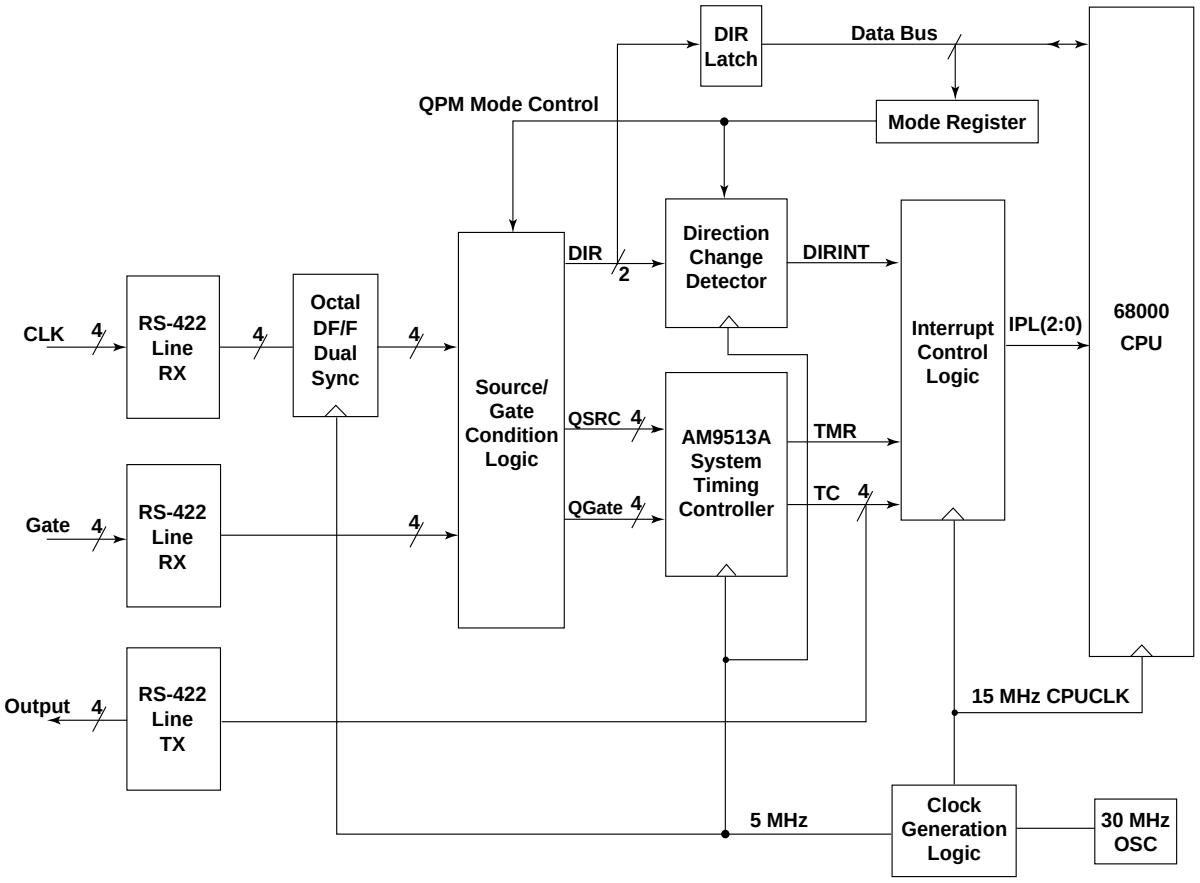


Figure 1-2 VMIVME-2540 System Timing Controller

The direction signal corresponding to each pair of even / odd channels is output from the source-gate conditioning logic to a latch (for the VMIVME-2540 local CPU to read) and to the QPM direction change interrupt logic. The QPM direction change logic will be discussed in a later section.

AM9513A System Timing Controller

The counting, timing, and waveform functions of the VMIVME-2540 are all controlled by the AM9513A STC. Each system timing controller has five very versatile 16-bit counter logic groups. Each counter logic group consists of a 16-bit counter, 16-bit load register, 16-bit hold register, and 16-bit counter mode register. In addition, counter logic groups one and two in the AM9513A have a 16-bit comparator and a 16-bit alarm register. Each 16-bit counter has programmable clocks and gate inputs from a number of sources, including the five clock and five gate inputs. Each counter logic group has a programmable output which may be configured to be inactive (high or low), terminal count (single clock cycle), or terminal count toggle (square wave). Counter logic groups may be cascaded to create higher precision 32-bit counters.

Of the five counter logic groups in the AM9513A, the VMIVME-2540 uses counters one through four to process the conditioned source and gate signals, while the fifth counter is used only for generating a periodic or delayed interrupts using the internal 5 MHz timebase. The internal timebase consists of a divider chain with four 4-bit

binary / BCD counters, providing a total of five clock sources for any of the counter logic groups. These five timebase frequency sources are used by the VMIVME-2540 to measure period and pulse width using one of five available clock frequencies (200 ns to 2 ms). Refer to the *AM9513A Technical Manual (1990 rev.) or the 1987 AMD MOS Microprocessor and Peripherals Handbook* for more application information on the system timing controller.

QPM Direction Change Interrupt Logic

The quadrature clock signals are synchronized to the 5 MHz STC clock by an octal register, and then the direction is detected in the source-gate conditioning logic. The direction signal goes to the QPM direction change interrupt logic, shown in Figure 1-3 below. The VMIVME-2540 local CPU uses the QPM direction change interrupt to interrogate the direction latch and record quadrature position profiles in VMEbus-accessible memory. The quadrature position measurement direction change interrupt logic is implemented using programmable logic.

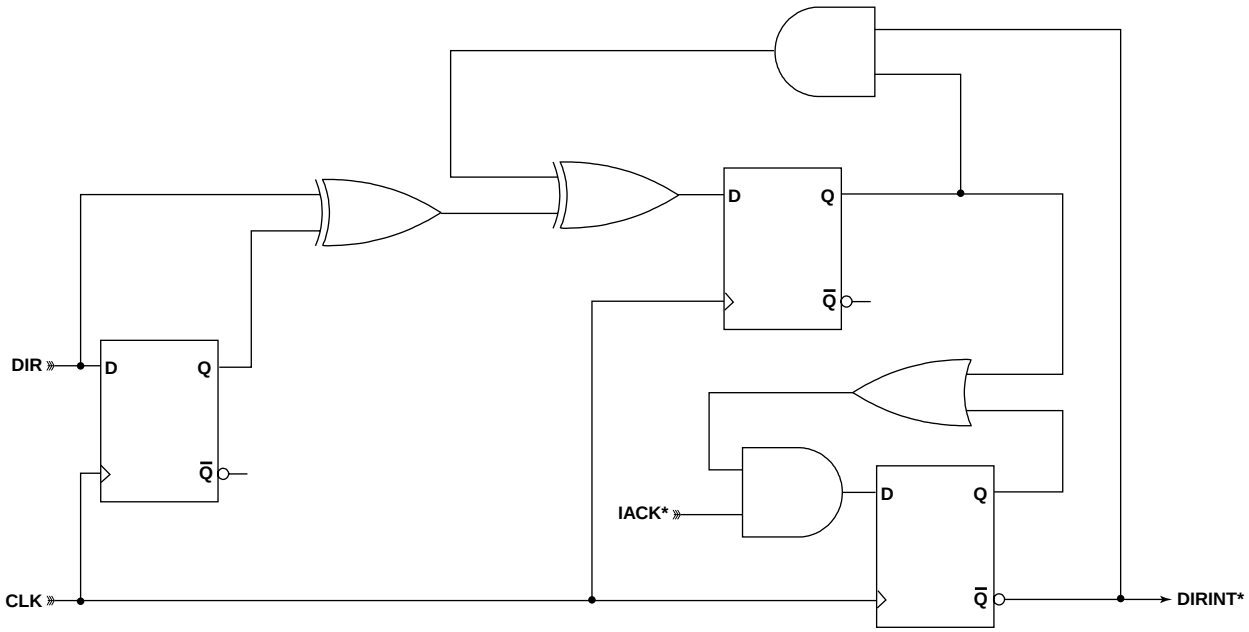


Figure 1-3 QPM Direction Change Interrupt Logic

I/O Processor

68HC000 CPU

The on-board intelligence of the VMIVME-2540 is provided by a 68HC000 microprocessor with a 15 MHz CPU clock. The HCMOS CPU has external 16-bit data/24-bit address buses, internal 32-bit data paths, and is a complex instruction set computer (CISC). Refer to *Motorola M68000 User's Manual*, 7th edition for more detail about the 68HC000.

Decode and Control Logic

All of the address decode, interrupt control, and arbitration functions for the local CPU are partitioned between two 68-pin PLDs. One PLD provides the following address decode and arbitration functions for the 68HC000:

- Local bus arbitration
- Local SRAM and EPROM chip selects
- Local I/O block decode and R/W strobes
- Data transceiver output controls
- User Command Interrupt/Command Status Buffer controls
- VMEbus DTACK and write strobe generation
- 15 MHz CPU clock and 5 MHz STC clock generation

The other PLD provides the following address decode and interrupt control functions:

- System timing controller chip selects decode
- QPM Mode Register and Direction Latch control strobes
- Local interrupt controller R/W strobes
- 68HC000 Interrupt Acknowledge bus cycle decode
- Local interrupt priority encoding - IPL(0:2)
- Configuration buffer /Status LED control
- Front panel reset /VMEbus reset combination into local reset signal

Local Bus Arbitration

The VMEbus may access a 64 Kbyte block of static RAM on the VMIVME-2540 board that is shared with the local 68HC000 CPU. This shared RAM and local data and address buses have two possible masters: the CPU has default control, while the VMEbus must request access. Figure 1-4 on page 25 shows the local resources arbitrated by PLD logic.

A bus request is issued to the 68HC000 whenever the following conditions are met:

a. A31:A24 matches the setting of DIP switch S2 (A32 mode only)

b. A23:A16 matches the setting of DIP switch S3

c. AM[5:0] matches the code for standard or extended data access

d. AM2 and jumpers E53/E54 match supervisory/nonprivileged access type

e. Signals IACK*, BGACK*, and SRBSEL* are not asserted.

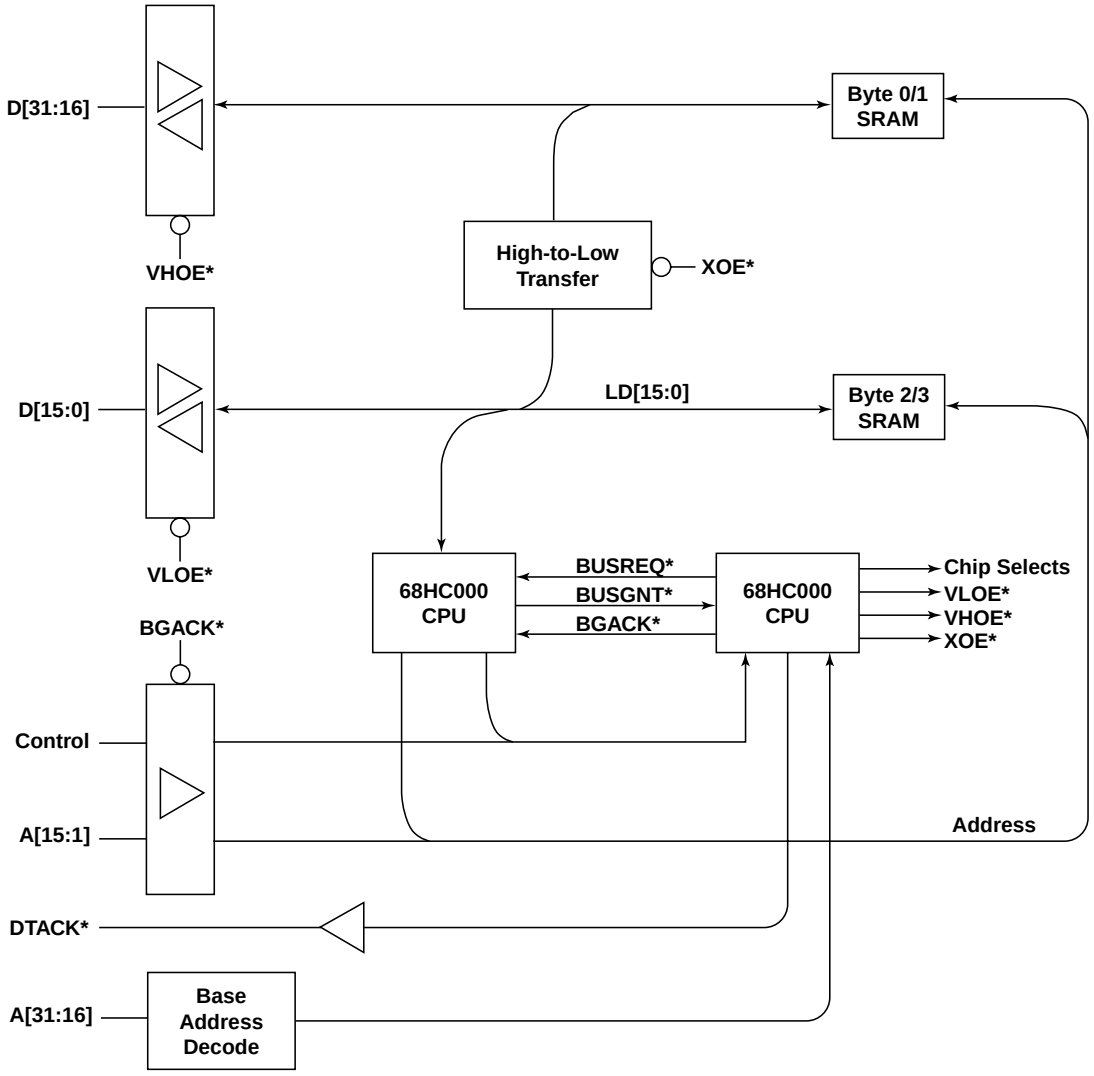


Figure 1-4 Local Bus Arbitration

The 68HC000 responds with a Bus Grant within 1.5 to 3.5 clock cycles after it completes the current bus cycle. One clock cycle after BUSGNT* is asserted, the local bus arbiter asserts Bus Grant Acknowledge (BGACK*), enabling the VMEbus buffers and local SRAM chip selects. VMEbus DTACK* is asserted synchronously to terminate the access, after which the VMEbus master withdraws control. Back-to-back and read-modify-write VMEbus accesses are not supported by the local bus arbiter: the 68HC000 executes its firmware in the same SRAM chips accessed by the VMEbus, thus the VMIVME-2540 local CPU is always granted the local bus after a VMEbus access.

Local Address Decode

The memory map of the board local resources occupies 1 Mbyte of the 16 Mbyte 68HC000 address space, thus only LA[19:1] are decoded. Table 1-1 below and Table 1-2 on page 27 list the local CPU and the local I/O address maps for the VMIVME-2540. The addresses in the tables are **not** accessible from the VMEbus; these are local resources accessed by the local CPU using those addresses. Note that for all local I/O addresses, three signals are decoded: IOSEL*, IORD*, and IOWR*. All local address decode applies to the VMIVME-2540 local CPU only. The VMEbus shared memory is accessible from both the local bus and the VMEbus; however, the local CPU address space used to access this memory has no correlation to the address space used for VMEbus access. This latter address space is user-controlled via switch settings.

Table 1-1 VMIVME-2540 Local CPU Address Map

HEX Address	Function
00000 to 0FFFF	EPROM Firmware
10000 to 7FFFF	Not Used
80000 to 8FFFF	I/O Address Space
90000 to DFFFF	Not Used
E0000 to EFFFF	Static RAM - local access only
F0000 to FFFFF	Static RAM – VMEbus-shared access

Table 1-2 VMIVME-2540 Local I/O Address Map

HEX Address	Mnemonic	Function	Comments
80000	RDCFG	Read board configuration straps	D14, D15
80000	WRENA	Write master interrupt enable to ID15	1 = enabled
80002	WRLED	Write status LED value to ID15	1 = on
80004	CLRCMD	Clear VMEbus command interrupt F/F	
80006	MODE	Quadrature Position Measurement Mode	Read / Write
80008	RDDIR	Read 16-bit QPM direction latch	
8000A	LDSRB	Load Slave Response Buffer (Command Status)	Write-only
80010	VA[3:0]	VMEbus Interrupt Vector 'A' low nibble	IIIC
80012	VA[7:4]	VMEbus Interrupt Vector 'A' high nibble	IIIC
80014	IRQA	VMEbus Interrupt 'A' IRQ level	IIIC
80018	VB[3:0]	VMEbus Interrupt Vector 'B' low nibble	IIIC
8001A	VB[7:4]	VMEbus Interrupt Vector 'B' high nibble	IIIC
8001C	IRQB	VMEbus Interrupt 'B' IRQ level	IIIC
80020	LTCRD	Read and Clear Interrupt Controller No.1 IRQs	TC0 - TC11
80022	LMASK	Write - Load Interrupt Controller No.1 mask	TC0 - TC11
80022	LMRD	Read - Read Interrupt Controller No.1 mask	TC0 - TC11
80024	UTCRD	Read and Clear Interrupt Controller No. 2 IRQs	TC12 - TC23
80026	UMASK	Write - Load Interrupt Controller No. 2 mask	TC12 - TC23
80026	UMRD	Read - Read Interrupt Controller No. 2 mask	TC12 - TC23
80028	TMRRD	Read and Clear Interrupt Controller No. 3 IRQs	TMR0 - TMR5
8002A	TMASK	Write - Load Interrupt Controller No. 3 mask	TMR0 - TMR5
8002A	TMRD	Read - Read Interrupt Controller No. 3 mask	TMR0 - TMR5
8002C	CLRDIR	Clear Direction Change Interrupt Controller	
80030 to 8003F	-	Not used.	
80040	STC0D	STC0 Data Port	
80042	STC0C	STC0 Control Port	
80044	STC1D	STC1 Data Port	
80046	STC1C	STC1 Control Port	
80048	STC2D	STC2 Data Port	
8004A	STC2C	STC2 Control Port	
8004C	STC3D	STC3 Data Port	
8004E	STC3C	STC3 Control Port	
80050	STC4D	STC4 Data Port	
80052	STC4C	STC4 Control Port	
80054	STC5D	STC5 Data Port	
80056	STC5C	STC5 Control Port	
80058 to 8FFFF	-	Not used.	

Local I/O Functions

The I/O functions accessed by the VMIVME-2540 local CPU are the AM9513A system timing controllers, the VMEbus interrupter modules and local QPM, and configuration and status LED interfaces.

a. **Configuration Buffer:** The VMIVME-2540 may have one, two, four, or six AM9513A system timing controllers, depending on the ordering option for the board. The number of system timing controllers present on the VMIVME-2540 board is encoded into configuration jumpers E50 and E51 as shown in Table 2-2 on page 41. The VMIVME-2540 local CPU reads these configuration jumpers with a buffer enabled by address $80000 during initialization.

b. **Master Interrupt Enable:** The local interrupt controller for the VMIVME-2540 local CPU may be completely disabled by writing a *zero* to address $80000. This master interrupt enable flip-flop is cleared by the RESET* signal and set by the VMIVME-2540 local CPU when initialization is complete.

c. **Status LED:** The front panel status LED is controlled by local I/O address $80002. The LED is illuminated when the VMIVME-2540 local CPU writes a logic one to this address and extinguished by a logic *zero*. The status LED is illuminated when the local CPU begins powerup/reset self-test, and extinguished when the self-test is completed. Thereafter, the LED will be illuminated if a failure or fatal error is detected. The processor is always halted and interrupts are disabled when a failure occurs and the LED is illuminated. The shared memory area remains accessible by the host for diagnostic information retrieval.

d. **Clear VMEbus Command Interrupt:** The VMEbus command interrupt flip-flop is cleared when the VMIVME-2540 local CPU asserts I/O address $80004.

e. **Quadrature Position Measurement Mode Register:** The QPM mode register has read/write access at I/O address $80006. Although the QPM mode register is implemented with two octal devices, only the least significant 12 bits are physically connected to the STC front-end logic.

f. **QPM Direction Latch:** The current direction of all quadrature channels may be read by the VMIVME-2540 local CPU at address $80008. As with the QPM Mode Register, this function is implemented with two octal latches, but only the least significant 12 bits contain direction information.

g. **Command Status Buffer:** The Command Status Buffer is an octal register located at local I/O address $8000A. Only the local VMIVME-2540 local CPU may write to this location. For the VMEbus host, this is a read-only buffer.

h. **Intelligent Interface Interrupt Controller (IIIC):** The VMIVME-2540 local CPU may assert VMEbus interrupts by writing vectors and IRQ levels to the Intelligent Interface Interrupt Controller, located at I/O addresses $80010-$80014 and $80018-$8001C. The IIIC has only a 4-bit read/write data interface to the VMIVME-2540 local CPU, requiring each value to be written in successive nibbles.

i. **Terminal Count Interrupt Controllers:** The system timing controller outputs TC[0:23] and TMR[0:5] comprise 30 interrupt sources to the VMIVME-2540 local CPU. These terminal count signals are logically arranged into three groups: signals TC[0:11] are conditioned by Interrupt Controller No.1, signals

TC[12:23] are handled by Interrupt Controller No. 2, and signals TMR[0:5] are combined and masked by Interrupt Controller No. 3. For each interrupt controller, the interrupt sources are captured into a register which may be read and cleared automatically by the VMIVME-2540 local CPU. Each interrupt controller also has a read / write mask register. An interrupt source is masked by a zero value and enabled by a logic one.

j. **System Timing Controller Ports:** The system timing controllers are addressed by the VMIVME-2540 local CPU as only two locations each: a control port and a data port. Transfers at the control port allow direct access to the STC command register when writing, and the STC status register when reading. All other STC internal locations are accessed through the data port with the desired location addressed by the STC data pointer register. Refer to the Advanced Micro Devices' **AM9513A/AM9513 Technical Manual** for more information about the system timing controller.

Local Memory

EPROM Firmware

The VMIVME-2540 local CPU retrieves its startup code and local interrupt vectors from two programmed EPROM ICs. The VMIVME-2540 local CPU fetches instructions and data from memory, word accesses (16 bits). Program execution is started in EPROM at powerup and, after RAM memory test is completed, code is copied to static RAM, program control is passed to RAM, and all program execution is from RAM from that point. The design of the LDTACK wait-state generator for the VMIVME-2540 local CPU dictates that the EPROMs have a data access time of 250 ns or less.

The EPROM firmware contains all programming to support the user programming of the board described in the “Programming” on page 49. The user controls the input/output operations performed by the system timing controllers using the command/command status interface and the channel control blocks, as described in Chapter 3 “Programming”.

Static RAM

The VMIVME-2540 local CPU has zero-wait-state access to 128 Kbyte of static RAM located in four 32 Kbyte x 8 ICs. As shown in Figure 1-4 on page 25, the VMIVME-2540 local CPU accesses this static RAM 16 bits at a time. For each longword in memory, bytes 2 and 3 connect directly to the VMIVME-2540’s local CPU 16-bit data bus, while bytes 0 and 1 are accessed through the data transfer transceivers. Strobes LWRSTB* and UWRSTB*, decoded from LDS* and UDS*, respectively, allow the 68HC000 to modify lower and upper bytes of static RAM independently. The design of the LDTACK wait-state generator for the VMIVME-2540 local CPU requires the use of static RAMs with a data access time of 45 ns or less for zero-wait-state operation.

A portion of the static RAM is interfaced to the VMEbus for sharing with the VMEbus host processor for control and communication. The VMEbus access to the static RAM may be either byte, word, or longword. The local bus arbitration logic only allows the VMEbus host to access the upper 64 Kbyte of the RAM, while the 68HC000 has granted the local bus. During the VMEbus access cycle, the assertion of RAM chip enables and write enables are synchronous to CPUCLK. VMEbus DTACK* is asserted one clock cycle after the rising edge of RAM write strobe during a VMEbus write cycle, ensuring the integrity of the written data.

Local Interrupt Controller

The local 68HC000 interrupt controller encodes 32 interrupt sources into seven prioritized interrupt request levels. STC terminal count signals TC[11:0] are grouped into 68HC000 IRQ level 6, signals TC[23:12] are grouped into IRQ level 5, and signals TMR[5:0] are grouped into IRQ level 4. The VMEbus command interrupt asserts IRQ level 7, while the quadrature direction change interrupt is mapped into 68HC000 IRQ level 3. IRQ levels 1 and 2 are not used in the VMIVME-2540 interrupt structure. The QPM direction change interrupt is described in “QPM Direction Change Interrupt Logic” on page 23. The following sections will describe STC and the VMEbus command interrupt sources.

STC Interrupts

Each system timing controller has five, 16-bit counters with programmable terminal count outputs. Each terminal count output may be programmed either to be high, low, a single clock pulse wide, or square wave output. In each STC, counters one through four are associated with the front panel I/O connectors and are members of the TC[23:0] bus, while the fifth counter output is intended only for creating a *soft* interrupt after a specified time and is a member of the TMR[5:0] bus. The STC terminal count interrupts are masked and edge-detected by 6-bit interrupt controllers, implemented via PLD logic. The logic inside the interrupt controller is shown in Figure 1-5 on page 32.

VMEbus Command Interrupt

The VMEbus asserts the command interrupt to the VMIVME-2540 local CPU by writing the command code to offset $0004 from the base address. PLD logic decodes this address and sets an internal flip-flop during the VMEbus command bus cycle. The VMIVME-2540 local CPU resets the command interrupt flip-flop by asserting local address $8000A.

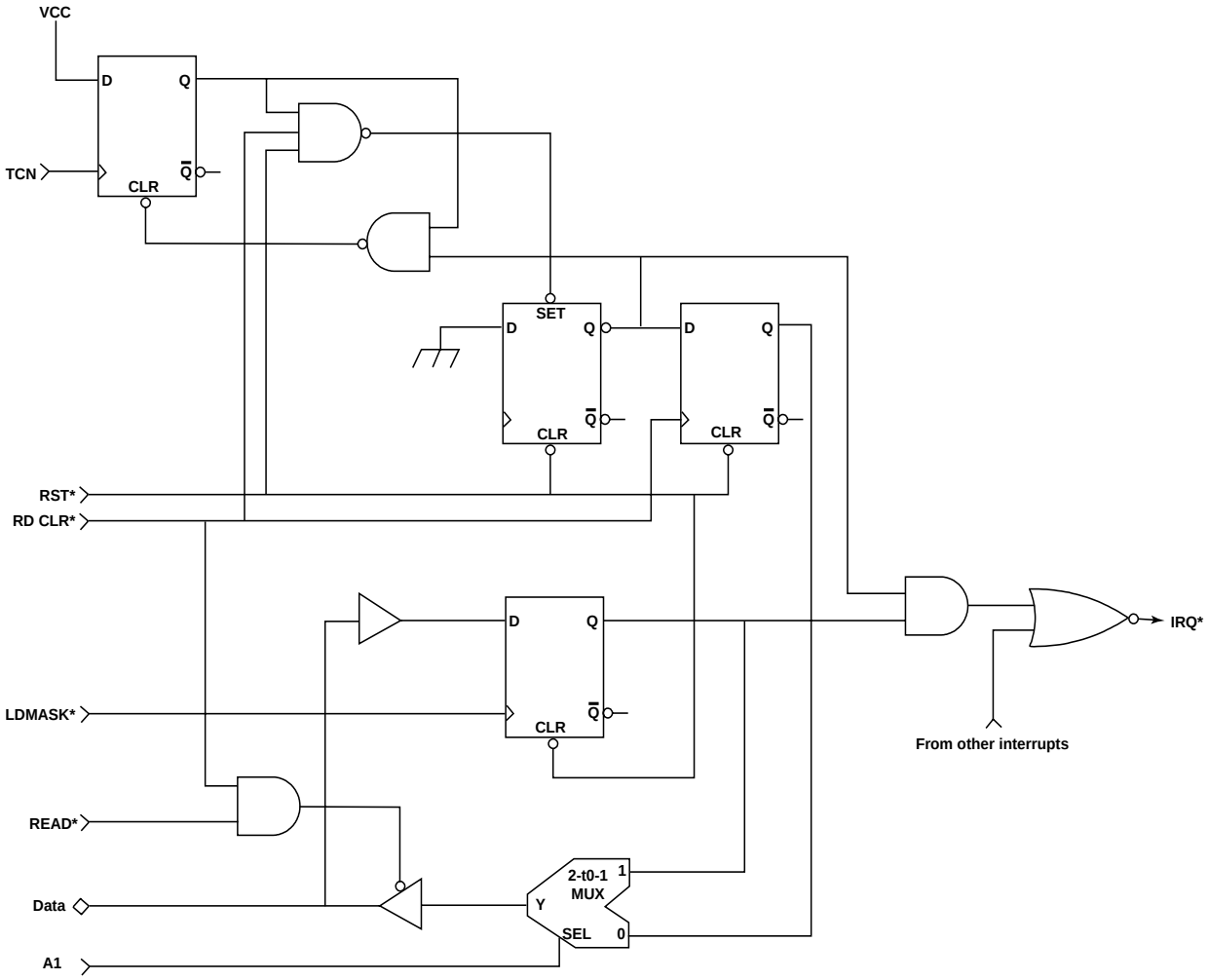


Figure 1-5 Interrupt Controller for STC Outputs

VMEbus Slave Interface

VMEbus Slave Address Decode

The “Local Bus Arbitration” on page 24 , was discussed in a general sense. Figure 1-6 on page 34 shows the actual VMEbus slave address decode logic and address snapshot registers on the VMIVME-2540 board. Comparators assert signals HCMP*, MCMP*, and LCMP* when the VMEbus address and address modifiers match the settings of DIP switches S1, S2, and the code for standard / extended data access.

The VMEbus specification requires that the address and control signals be stable for at least 10 ns prior to the assertion of DS0* / DS1*, and held valid a minimum of 30 ns after this assertion. DS0* / DS1* is combined to create the signal VSTB, the composite active-high VMEbus data strobe. A sufficient time after the rising edge of VSTB, the signal VACLK takes a snapshot of the VMEbus address comparison signals HCMP*, MCMP*, and LCMP*, control signals LWORD*, WRITE*, AM2, and the lower VMEbus address bus. Thus, all of the VMEbus lower address control, and board select information is held constant until the next falling edge of DS0* / DS1*.

A delay line is used to shorten the leading edge of VSTB* so that all registered address and control information is valid going into the local bus arbiter. Once VSTB* is asserted and all other comparison information is valid, the PLD logic asserts BUSREQ* asynchronously to the VMIVME-2540 local CPU (the 68HC000), and the VMEbus access proceeds as described in “Local Bus Arbitration” on page 24.

Command Status Code

When the VMIVME-2540 local CPU has finished processing a commanded action, a command status code is placed in the command status buffer. The VMEbus host can poll this register with minimal delay since it is not arbitrated as a local resource. This command status buffer benefits the VMIVME-2540 local CPU by increasing the portion of the local bus bandwidth dedicated to the VMIVME-2540 local CPU, and not to VMEbus access service. The address decode for the VMEbus polling buffer occurs in two parts: the upper 16 address lines by one PLD device, while the lower sixteen address lines are decoded a second PLD. The second device generates signal SRBSEL* which is connected to the first PLD device. The first PLD then generates signal SRBOE* if the VMEbus data access address is offset $0006 from the base address. Signal SRBOE* enables the status code data onto the VMEbus data bus lines VLD[7:0] and also asserts a VMEbus DTACK* through the VMEbus interrupter module described in the next section. The VMEbus DTACK* delay from decode of signal SRBOE* is one to two 15 MHz clock periods.

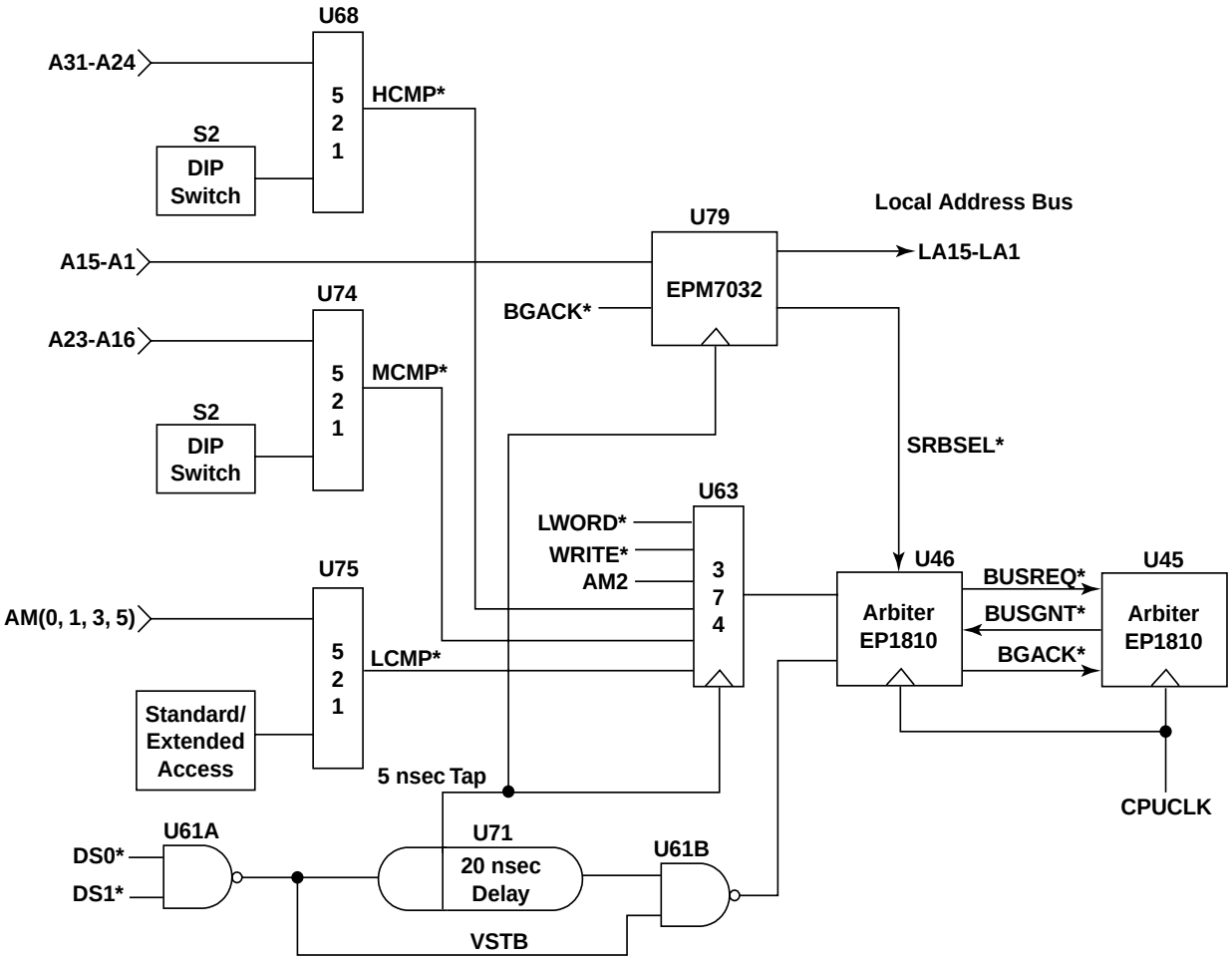


Figure 1-6 VMEbus Slave Address Decode

VMEbus Interrupter Modules

The VMIVME-2540 has two interrupt modules implemented in PLD. Figure 1-7 below shows the block diagram of the Intelligent Interface Interrupter Module. This design allows the 68HC000 microprocessor to interrupt a VMEbus master on any two of the seven IRQ levels simultaneously without requiring the arbitration for the local bus during the IACK cycle.

The VMIVME-2540 local CPU programs the interrupter module through a 4-bit bidirectional data port by asserting CS*, RDWR* and the 3-bit address of the nibble to be written. Vectors A and B are 8-bit registers which contain the VMEbus interrupt vectors. ENA / IRQA and ENA / IRQB are 4-bit registers which contain the 3-bit VMEbus IRQ level to be asserted and IRQ enable. The status of these registers may be verified at any time by the VMIVME-2540 local CPU through the same local data port. Refer to Table 1-2 on page 27 for address decode information. Note that writing the code 8 to either IRQ / enable control location does not enable a VMEbus IRQ level (there is no IRQ 0).

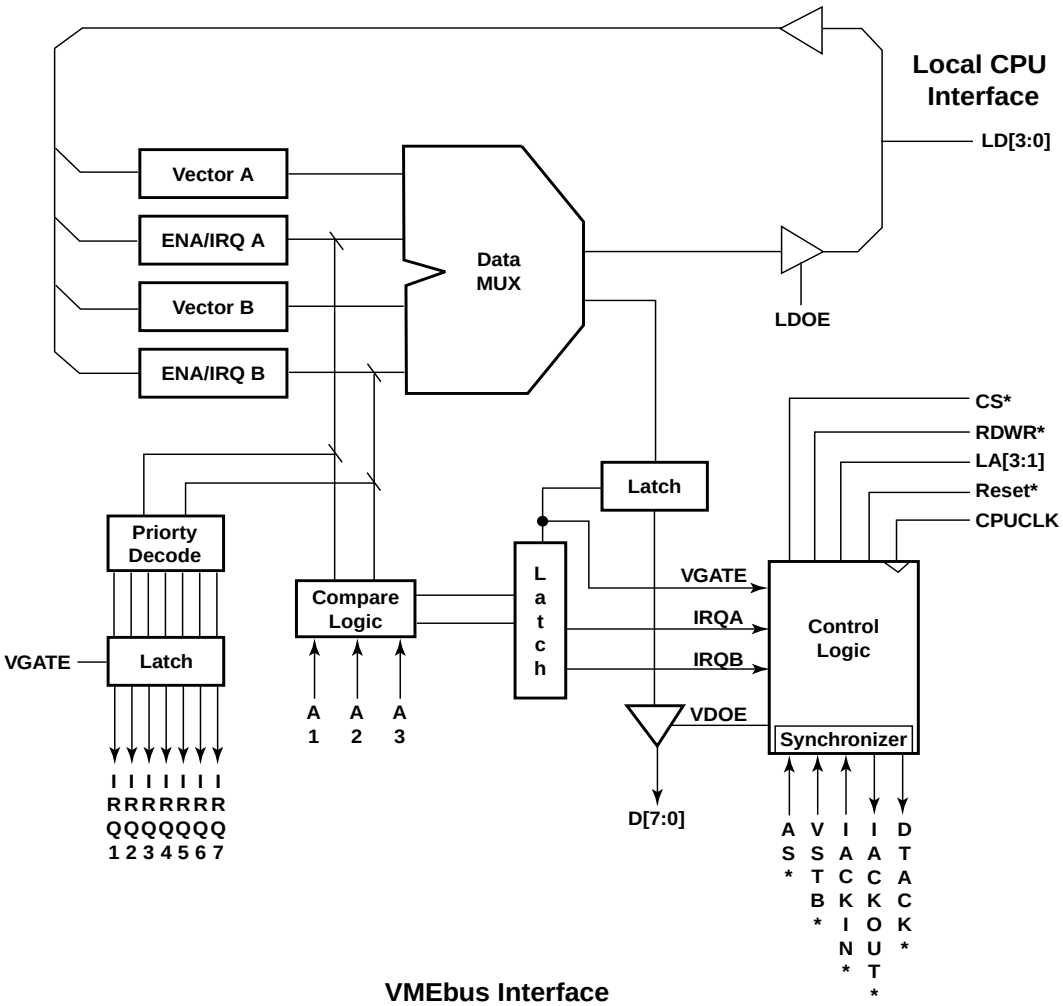


Figure 1-7 Intelligent Interface Interrupter Module

CHAPTER
2

Configuration and Installation

Contents

Unpacking Procedures 38
Configuration 39
I/O Connector Pin Assignments 42
Recommended Discrete Wire Connectors and Terminal Blocks 45
TTL/Single-Ended Input Signal Compatibility Configuration 46

Unpacking Procedures

CAUTION: Some of the components assembled on VMIC's products may be sensitive to electrostatic discharge and damage may occur on boards that are subjected to a high-energy electrostatic field. When the board is placed on a bench for configuring, etc., it is suggested that conductive material should be inserted under the board to provide a conductive shunt. Unused boards should be stored in the same protective boxes in which they were shipped.

Upon receipt, any precautions found in the shipping container should be observed. All items should be carefully unpacked and thoroughly inspected for damage that might have occurred during shipment. The board(s) should be checked for broken components, damaged printed circuit board(s), heat damage, and other visible contamination. All claims arising from shipping damage should be filed with the carrier and a complete report sent to VMIC together with a request for advice concerning the disposition of the damaged item(s).

Configuration

The VMIVME-2540 printed circuit board has two 8-position DIP switches and several jumper options, as shown in Figure 2-1 below. The configurations of these options are discussed in the following sections.

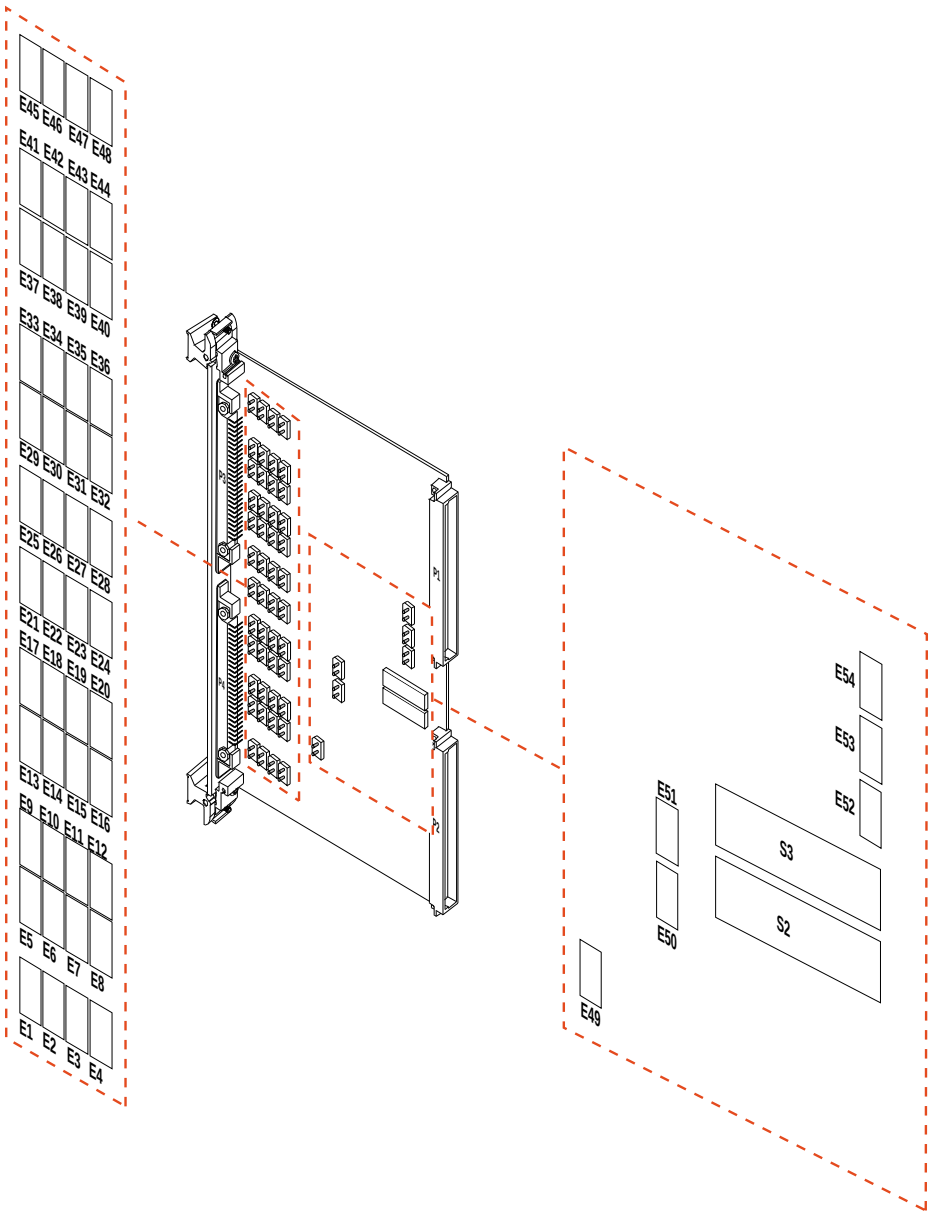


Figure 2-1 VMIVME-2540 DIP Switches and Jumper Options

DIP Switch Settings

The VMEbus base address of the VMIVME-2540 is set by 8-position DIP switches S2 and S3. The settings of DIP switch S2 correspond to VMEbus address bits A31 through A24, while the settings of DIP switch S3 correspond to VMEbus address bits A23 through A16. Setting an individual switch to the **on** position matches a *zero* address bit value, conversely, a *one* address bit value compares to the **off** position. Figure 2-2 below shows the settings for DIP switches S2 and S3 for a sample VMEbus address range.

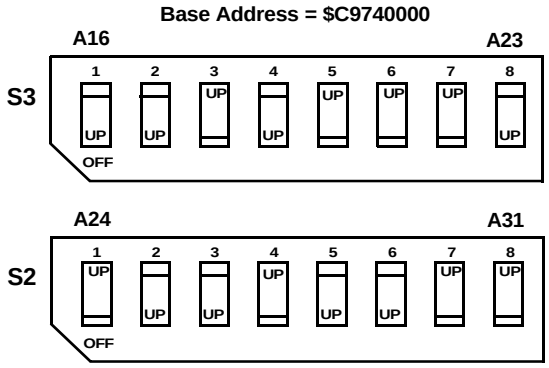


Figure 2-2 Example DIP Switch Settings

Jumper Options

The VMIVME-2540 printed circuit board has 51 user-configurable and three fixed jumper options as shown in Figure 2-1 on page 39. Forty-eight of the user-configurable jumper options (E1 through E48) allow the user to remove the RS-422 receiver termination resistors for TTL compatibility. Table 2-1 below shows the correspondence between jumpers E1-E48 and channel 0 to 23 clock and gate receiver termination resistors.

NOTE: Proper RS-422 line termination uses only one 120 termination resistor. Line driver damage can result from driving more than one termination resistor.

Table 2-1 RS-422 Receiver Termination Jumper Options

Input	Jumper	Input	Jumper	Input	Jumper	Input	Jumper
CLOCK0	E28	CLOCK6	E40	CLOCK12	E4	CLOCK18	E16
GATE0	E27	GATE6	E39	GATE12	E3	GATE18	E15
CLOCK1	E26	CLOCK7	E38	CLOCK13	E2	CLOCK19	E14
GATE1	E25	GATE7	E37	GATE13	E1	GATE19	E13
CLOCK2	E32	CLOCK8	E44	CLOCK14	E8	CLOCK20	E20
GATE2	E31	GATE8	E43	GATE14	E7	GATE20	E19
CLOCK3	E30	CLOCK9	E42	CLOCK15	E6	CLOCK21	E18
GATE3	E29	GATE9	E41	GATE15	E5	GATE21	E17
CLOCK4	E36	CLOCK10	E48	CLOCK16	E12	CLOCK22	E24
GATE4	E35	GATE10	E47	GATE16	E11	GATE22	E23
CLOCK5	E34	CLOCK11	E46	CLOCK17	E10	CLOCK23	E22
GATE5	E33	GATE11	E45	GATE17	E9	GATE23	E21

Jumper option E49 corresponds to the type of EPROMs installed on the VMIVME-2540 board and is installed at the factory.

Jumper options E51 and E50 correspond to the VMIVME-2540 ordering options and are installed by the factory. Table 2-2 below list the placement of wire jumpers E50 and E51 and the corresponding configurations.

Table 2-2 Configuration Jumpers E50 and E51

Dash	E50	E51	Configuration
-000	Installed	Installed	Four Channels
-100	No Jumper	Installed	Eight Channels
-200	Installed	No Jumper	Sixteen Channels
-300	No Jumper	No Jumper	Twenty-Four Channels

Jumper option E52 controls whether the VMIVME-2540 responds to standard VMEbus address space (A24) or extended VMEbus address space (A32). With the shunt installed at location E52, the VMIVME-2540 is configured as an A32 VMEbus slave. When location E52 does not have a shunt installed, the board is configured as an A24 VMEbus slave.

Jumper options E53 and E54 determine whether the VMIVME-2540 will occupy supervisory only, nonprivileged only, or both supervisory and nonprivileged VMEbus address spaces. Table 2-3 below list the placements of jumper E52, E53 and E54 with the corresponding responses of the VMIVME-2540 board.

Table 2-3 VMEbus Access Select Jumpers E52, E53 and E54

VMEbus Access Type	E52	E53	E54
Supervisory Only-Standard / A24	No Jumper	Installed	No Jumper
Nonprivileged Only - Standard / A24	No Jumper	Installed	Installed
Supervisory Only - Extended / A32	Installed	Installed	No Jumper
Nonprivileged Only - Extended / A32	Installed	Installed	Installed
Supervisory and Nonprivileged - Standard / A24	No Jumper	No Jumper	No Jumper
Supervisory and Nonprivileged - Extended / A32	Installed	No Jumper	No Jumper
VMEbus Response Disabled	Don’t Care	No Jumper	Installed

I/O Connector Pin Assignments

The VMIVME-2540 front panel connector pin assignments are shown in Table 2-4 below for P3 and Table 2-5 on page 43 for P4. Connector orientation is shown in Figure 2-3 on page 44. P3 and P4 are DIN 41612 Type C 96-pin male connectors with multiple fixing brackets. Each connector supports 12 measurement/control channels consisting of differential clock, gate, and output signals as well as ground and TTL-compatibility voltage (VTTL) pins. The VTTL voltage is a nominal 1.4 V source which may be externally tied to the inverting input of the RS-422 line receivers for TTL input compatibility.

Table 2-4 I/O Connector P3 Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal
A32	CLK11-	B32	GATE11-	C32	OUT11-
A31	CLK11+	B31	GATE11+	C31	OUT11+
A30	VTTL	B30	GND	C30	VTTL
A29	CLK10-	B29	GATE10-	C29	OUT10-
A28	CLK10+	B28	GATE10+	C28	OUT10+
A27	GND	B27	VTTL	C27	GND
A26	CLK9-	B26	GATE9-	C26	OUT9-
A25	CLK9+	B25	GATE9+	C25	OUT9+
A24	CLK8-	B24	GATE8-	C24	OUT8-
A23	CLK8+	B23	GATE8+	C23	OUT8+
A22	VTTL	B22	GND	C22	VTTL
A21	CLK7-	B21	GATE7-	C21	OUT7-
A20	CLK7+	B20	GATE7+	C20	OUT7+
A19	GND	B19	VTTL	C19	GND
A18	CLK6-	B18	GATE6-	C18	OUT6-
A17	CLK6+	B17	GATE6+	C17	OUT6+
A16	CLK5-	B16	GATE5-	C16	OUT5-
A15	CLK5+	B15	GATE5+	C15	OUT5+
A14	VTTL	B14	GND	C14	VTTL
A13	CLK4-	B13	GATE4-	C13	OUT4-
A12	CLK4+	B12	GATE4+	C12	OUT4+
A11	GND	B11	VTTL	C11	GND
A10	CLK3-	B10	GATE3-	C10	OUT3-
A9	CLK3+	B9	GATE3+	C9	OUT3+
A8	CLK2-	B8	GATE2-	C8	OUT2-
A7	CLK2+	B7	GATE2+	C7	OUT2+
A6	VTTL	B6	GND	C6	VTTL
A5	CLK1-	B5	GATE1-	C5	OUT1-
A4	CLK1+	B4	GATE1+	C4	OUT1+
A3	GND	B3	VTTL	C3	GND
A2	CLK0-	B2	GATE0-	C2	OUT0-
A1	CLK0+	B1	GATE0+	C1	OUT0+

Table 2-5 I/O Connector P4 Pin Assignments

PIN	SIGNAL	PIN	SIGNAL	PIN	SIGNAL
A32	CLK23-	B32	GATE23-	C32	OUT23-
A31	CLK23+	B31	GATE23+	C31	OUT23+
A30	VTTL	B30	GND	C30	VTTL
A29	CLK22-	B29	GATE22-	C29	OUT22-
A28	CLK22+	B28	GATE22+	C28	OUT22+
A27	GND	B27	VTTL	C27	GND
A26	CLK21-	B26	GATE21-	C26	OUT21-
A25	CLK21+	B25	GATE21+	C25	OUT21+
A24	CLK20-	B24	GATE20-	C24	OUT20-
A23	CLK20+	B23	GATE20+	C23	OUT20+
A22	VTTL	B22	GND	C22	VTTL
A21	CLK19-	B21	GATE19-	C21	OUT19-
A20	CLK19+	B20	GATE19+	C20	OUT19+
A19	GND	B19	VTTL	C19	GND
A18	CLK18-	B18	GATE18-	C18	OUT18-
A17	CLK18+	B17	GATE18+	C17	OUT18+
A16	CLK17-	B16	GATE17-	C16	OUT17-
A15	CLK17+	B15	GATE17+	C15	OUT17+
A14	VTTL	B14	GND	C14	VTTL
A13	CLK16-	B13	GATE16-	C13	OUT16-
A12	CLK16+	B12	GATE16+	C12	OUT16+
A11	GND	B11	VTTL	C11	GND
A10	CLK15-	B10	GATE15-	C10	OUT15-
A9	CLK15+	B9	GATE15+	C9	OUT15+
A8	CLK14-	B8	GATE14-	C8	OUT14-
A7	CLK14+	B7	GATE14+	C7	OUT14+
A6	VTTL	B6	GND	C6	VTTL
A5	CLK13-	B5	GATE13-	C5	OUT13-
A4	CLK13+	B4	GATE13+	C4	OUT13+
A3	GND	B3	VTTL	C3	GND
A2	CLK12-	B2	GATE12-	C2	OUT12-
A1	CLK12+	B1	GATE12+	C1	OUT12+

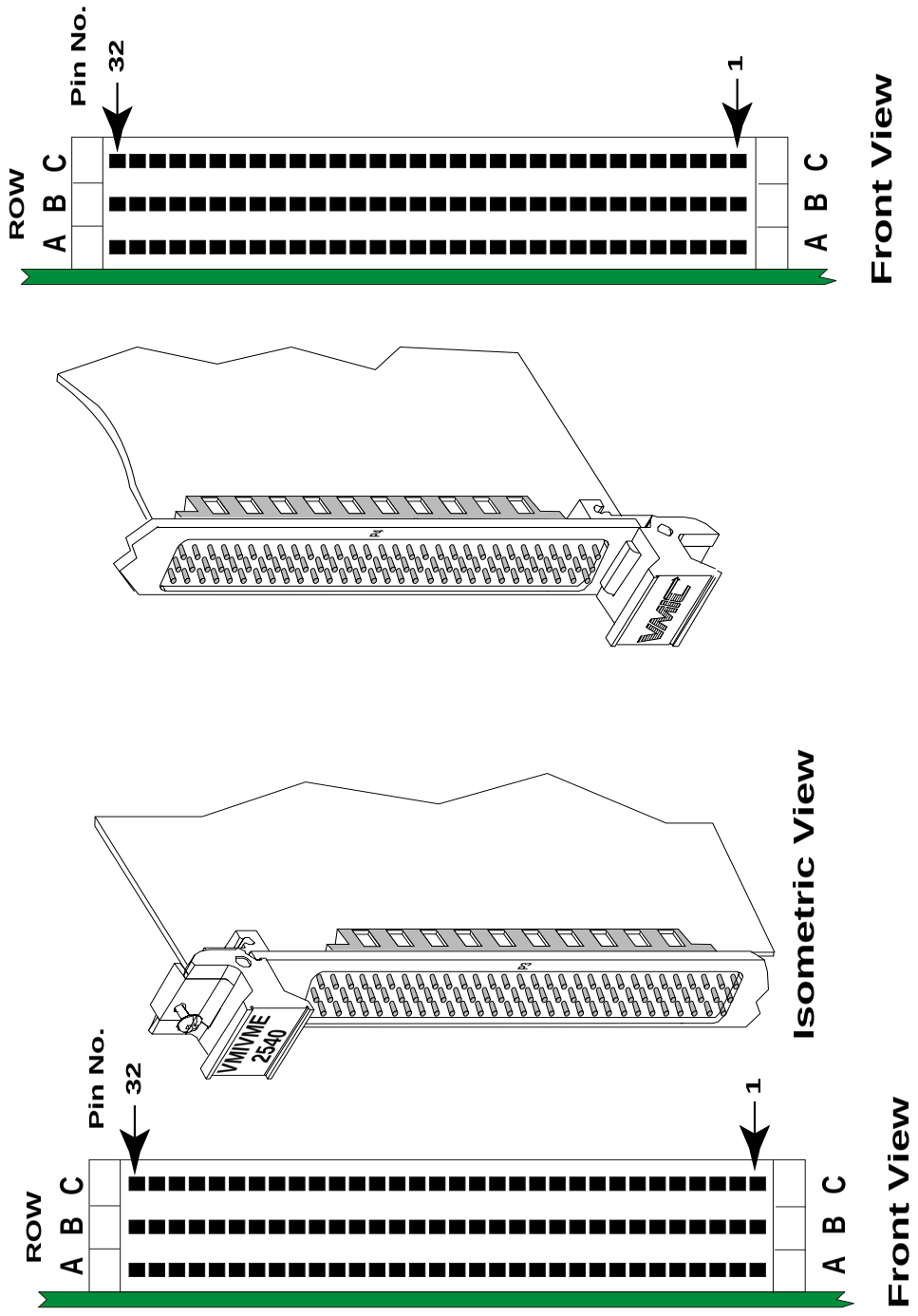


Figure 2-3 P3/P4 Pinout Layout

Recommended Discrete Wire Connectors and Terminal Blocks

The VMIVME-2540 has a large amount of front panel I/O which must be broken out by cabling and possibly rack-mounted terminal blocks. If the user wishes to use discrete wire connectors and cables, the suggested manufacturer is *Harting Elektronik, Inc., 2155 Stonington Ave., Suite 212, P.O. Box 95710, Hoffman Estates, Illinois 60195-0710*. The recommended connector components for cabling to P3 and P4 are given in the table below.

Table 2-6 Recommended Discrete Wire Connectors and Accessories

COMPONENT	HARTING CATALOG #
96-pin Discrete Wire Connector	0903-096-3214
Female Crimp Contacts	0902-000-8484
Connector Shell Housing	0903-096-0501
Left-Locking Lever	0902-000-9902
Right-Locking Lever	0902-000-9903

The RS-422 differential signals should be connected by twisted-pair insulated wires, 24 AWG solid or stranded copper conductors (R < 30/1,000 ft for other conductors) with a maximum cable length of 4,000 feet. Care should be taken to ensure that each signal group is properly grounded for reliable operation.

TTL signals may be connected by flat 96-conductor ribbon cable, 30 AWG insulated copper-stranded conductor. The corresponding female 96-pin DIN connector for flat-ribbon cable is ERNI 913.031 or similar. It is recommended that the total cable length be 50 feet or less for the TTL application with attention given to maximum signal transition rate and signal degradation over distance. RS-422 differential interface is clearly superior for longer cable lengths.

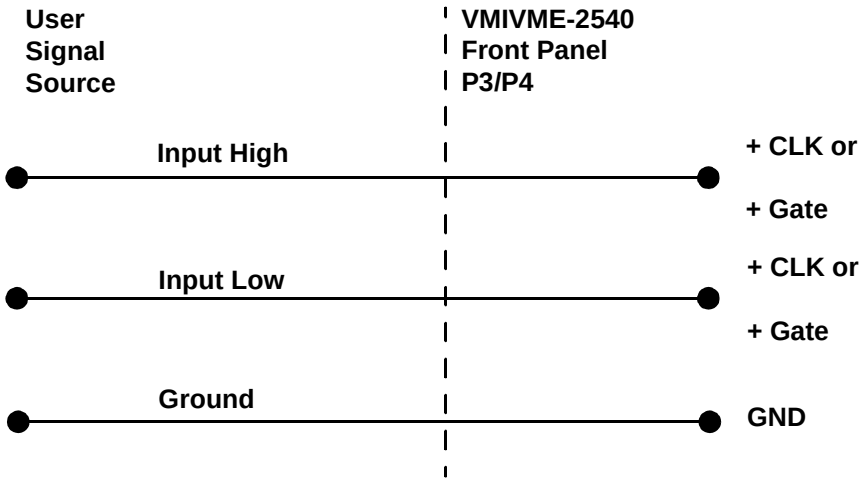
If the user wishes the I/O signals from the VMIVME-2540 front panel to be broken out at terminal blocks, the VMIACC-BT04 Dual 96-pin Transition Panel is suggested. The VMIACC-BT04 is a 19-inch rack-mountable panel which breaks all 192 signals out from two DIN 96-pin connectors to standard terminal blocks. Connection between the VMIACC-BT04 and the VMIVME-2540 front panel is made by 96-conductor ribbon cables, three foot length recommended.

TTL/Single-Ended Input Signal Compatibility Configuration

The VMIVME-2540 can be configured for single-ended input signals on a channel-by-channel basis by removing jumper shunts and externally wiring the TTL compatibility voltage (VTTL) to the corresponding inverting input of the RS-422 line receiver (inputs with a (-) suffix). Connection of the VTTL input to the (-) input will establish the VTTL voltage level as the threshold for the single-ended input signal at the (+) input. This input signal is connected to the noninverting input of the RS-422 line receiver (inputs with a (+) suffix). Care must be taken to limit the voltage range of the input signal when the jumper shunts are removed. Also, exercise caution in the connection of the VTTL signal to the (-) input(s) to avoid damage to the VTTL voltage generation circuitry of the VMIVME-2540. Select the channels which are to be configured for TTL operation, remove the associated gate and clock jumpers for those channels. Make the associated wiring changes for VTTL-to-input on the P3 and / or P4 front panel connectors.Table 2-7 below is the TTL / Single-Ended signal configuration. Figure 2-4 on page 47 is a typical RS-422 Signal connection and Figure 2-5 is the typical TTL signal connection.

Table 2-7 TTL/Single-Ended Input Signal Compatibility Configuration

Clock Input	Remove Jumper	Gate Input	Remove Jumper	VTTL Feedback	Ground Reference
CLOCK0	E28	GATE0	E27	P3-B3	P3-A3
CLOCK1	E26	GATE1	E25	P3-A6	P3-C3
CLOCK2	E32	GATE2	E31	P3-C6	P3-B6
CLOCK3	E30	GATE3	E29	P3-B11	P3-A11
CLOCK4	E36	GATE4	E35	P3-A14	P3-C11
CLOCK5	E34	GATE5	E33	P3-C14	P3-B14
CLOCK6	E40	GATE6	E39	P3-B19	P3-A19
CLOCK7	E38	GATE7	E37	P3-A22	P3-C19
CLOCK8	E44	GATE8	E43	P3-C22	P3-B22
CLOCK9	E42	GATE9	E41	P3-B27	P3-A27
CLOCK10	E48	GATE10	E47	P3-A30	P3-C27
CLOCK11	E46	GATE11	E45	P3-C30	P3-B30
CLOCK12	E4	GATE12	E3	P4-B3	P4-A3
CLOCK13	E2	GATE13	E1	P4-A6	P4-C3
CLOCK14	E8	GATE14	E7	P4-C6	P4-B6
CLOCK15	E6	GATE15	E5	P4-B11	P4-A11
CLOCK16	E12	GATE16	E11	P4-A14	P4-C11
CLOCK17	E10	GATE17	E9	P4-C14	P4-B14
CLOCK18	E16	GATE18	E15	P4-B19	P4-A19
CLOCK19	E14	GATE19	E13	P4-A22	P4-C19
CLOCK20	E20	GATE20	E19	P4-C22	P4-B22
CLOCK21	E18	GATE21	E17	P4-B27	P4-A27
CLOCK22	E24	GATE22	E23	P4-A30	P4-C27
CLOCK23	E22	GATE23	E21	P4-C30	P4-B30



NOTE: *Clock (CLK) or Gate input connections vary depending on the measurements to be performed. See Chapter 3 “Programming” for details.

Figure 2-4 Typical RS-422-Compatible Signal Connections

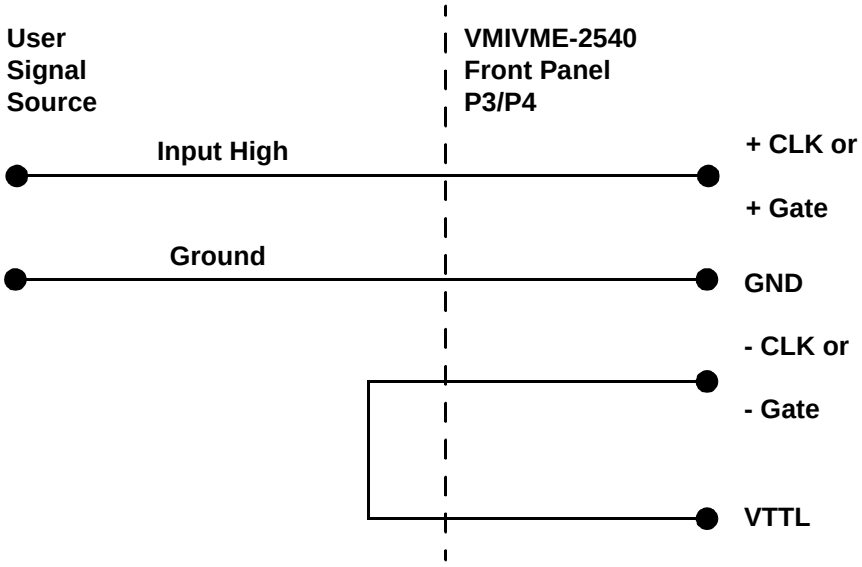


Figure 2-5 Typical TTL-Compatible Signal Connections

NOTE: *Clock (CLK) or Gate input connections vary depending on the measurements to be performed. See Chapter 3 “Programming” for details.

CHAPTER

3

Programming

Contents

VMEbus Interface Memory Map 51
Command Interface 60
Command Status Codes 63
Modes of Operation 64
Format of the Operation Mode Select Flag: 69
Command Descriptions 70
Getting Started 102

Introduction

The user controls the operation of the VMIVME-2540 via the local CPU through a shared memory interface, which provides a command-driven system. The shared memory is a 64 Kbyte contiguous block of memory and is structured within the local CPU program to satisfy the definitions shown in Table 3-1 on page 51. This structure definition provides the command register, command status reporting, measurement interface, and I/O channel control blocks (CCBs) for the exchange of data between the local CPU and the user's host processor.

Input and output operations are initiated by writing the parameters for the operation to the desired CCB, selecting a channel via the channel select register (byte offset $000A) and sending the associated command to the local CPU by writing the command to the command register (word at offset $0004; byte offset at $0005), and monitoring the command status buffer (word at offset $0006; byte offset at $0007).

The command word at offset $0004 and the command status buffer at offset $0006 have special attributes and handling properties for the local CPU and are the primary synchronization method for the VMEbus host program. When the VMEbus host writes a command word to offset $0004 (or byte to offset $0005) from the base address, an interrupt is generated for the VMIVME-2540 local CPU.

The VMIVME-2540 local CPU responds to the command interrupt by reading the command and channel ID locations, performs the commanded action defined by the command and the parameters in the CCB of the selected channel, and writes the command status code to the command status buffer. The command status buffer is a read-only register and cannot be cleared by writing a zero to the register. The command status register can be cleared by writing the clear command status command code ($1C) to the command register, which will cause the local processor to clear the command status byte. The host should verify that this register is clear (zero) prior to initiating a command.

The VMEbus host can poll the command status buffer to wait for a nonzero response code, or the command status IRQ and command status vector locations can be initialized with a VMEbus IRQ level and vector, and host interrupt processing used for command status reporting.

The VMIVME-2540 performs a variety of the input and output operations under command control of the host processor. Input operations for signal measurement processes can be interrupt-driven, polled through the measurement queue interface, polled via the channel status buffer of the CCB, or polled via the channel's data validity flag.

Channels can also be set up to measure continuously through the continuous-discrete descriptor (offset $000B from the base address). The VMEbus-shared memory addresses, command codes, command status codes, and channel control blocks entries will be discussed in the following sections.

VMEbus Interface Memory Map

The memory map for the VMIVME-2540 resources, accessible by the VMEbus host is given in Table 3-1. The contents of the shared memory area are initialized to zero (0) at powerup / reset / initialization unless otherwise noted in the following descriptions.

NOTE: The following formats are presented in terms of the vmebus with byte accesses shown in terms of the data bus lines on which they are accessed; that is, odd bytes are accessed on bus data lines D0 through D7, and even bytes are accessed on data lines D8 through D15. All byte-width data, once moved to the internal registers of most host processors, will be bit-accessed using Bit 0 through Bit 7 for bit manipulation operations.

Table 3-1 VMIVME-2540 VMEbus I/F Addresses

Offset	Function	Access	Size
$0000	Board ID / Configuration	*Read / Write	Word
$0002	Firmware Revision Level	*Read / Write	Word
$0004	Command	Read / Write	Word / Byte***
$0006	Command Status	Read	Word / Byte***
$0008	Command Status IRQ Level	Read / Write	Byte
$0009	Command Status Interrupt Vector	Read / Write	Byte
$000A	Channel ID	Read / Write	Byte
$000B	Continuous / Discrete Flag	Read / Write	Byte
$000C	Measurement Flag	Read / Write	Byte
$000E	Measurement Channel	Read / Write	Byte
$000F	Measurement Status	Read / Write	Byte
$0010	Ch 0 Channel Control Block	Read / Write	**See CCB DESC
$0020	Ch 1 Channel Control Block	Read / Write	**See CCB DESC
$0030	Ch 2 Channel Control Block	Read / Write	**See CCB DESC
$0040	Ch 3 Channel Control Block	Read / Write	**See CCB DESC
$0050	Ch 4 Channel Control Block	Read / Write	**See CCB DESC
$0060	Ch 5 Channel Control Block	Read / Write	**See CCB DESC
$0070	Ch 6 Channel Control Block (1)	Read / Write	**See CCB DESC
$0080	Ch 7 Channel Control Block (1)	Read / Write	**See CCB DESC
$0090	Ch 8 Channel Control Block (2)	Read / Write	**See CCB DESC
$00A0	Ch 9 Channel Control Block (2)	Read / Write	**See CCB DESC
$00B0	Ch 10 Channel Control Block (2)	Read / Write	**See CCB DESC
$00C0	Ch 11 Channel Control Block (2)	Read / Write	**See CCB DESC
$00D0	Ch 12 Channel Control Block (2)	Read / Write	**See CCB DESC
$00E0	Ch 13 Channel Control Block (2)	Read / Write	**See CCB DESC
$00F0	Ch 14 Channel Control Block (2)	Read / Write	**See CCB DESC
$0100	Ch 15 Channel Control Block (2)	Read / Write	**See CCB DESC
$0110	Ch 16 Channel Control Block (3)	Read / Write	**See CCB DESC
$0120	Ch 17 Channel Control Block (3)	Read / Write	**See CCB DESC
$0130	Ch 18 Channel Control Block (3)	Read / Write	**See CCB DESC
$0140	Ch 19 Channel Control Block (3)	Read / Write	**See CCB DESC

Table 3-1 VMIVME-2540 VMEbus I/F Addresses (Continued)

Offset	Function	Access	Size
$0150	Ch 20 Channel Control Block (3)	Read / Write	**See CCB DESC
$0160	Ch 21 Channel Control Block (3)	Read / Write	**See CCB DESC
$0170	Ch 22 Channel Control Block (3)	Read / Write	**See CCB DESC
$0180	Ch 23 Channel Control Block (3)	Read / Write	**See CCB DESC
$0190	Timer 0 Control Block	Read / Write	***See TCCB DESC
$01A0	Timer 1 Control Block (1)	Read / Write	***See TCCB DESC
$01B0	Timer 2 Control Block (2)	Read / Write	***See TCCB DESC
$01C0	Timer 3 Control Block (2)	Read / Write	***See TCCB DESC
$01D0	Timer 4 Control Block (3)	Read / Write	***See TCCB DESC
$01E0	Timer 5 Control Block (3)	Read / Write	***See TCCB DESC
$01F0-$0207	Continuous-Discrete Flags	*Read / Write	Byte
$0208-$021F	Measurement Data Validity Flags	Read / Write	Byte
$0220-$03DF	Internal Flags / Reserved	*Read / Write	Byte / Word / Long
$03E0-$03FF	Firmware Release Information	*Read / Write	Byte / Word / Long
$0400-$3FFF	Diagnostic / Utility Buffer	*Read / Write	Byte / Word / Long
$4000-$FFFF	Scratch Pad / Reserved	*Read / Write	Byte / Word / Long

NOTE: * User should not write to this area: value initialized / used by local CPU.
** CCB is the Channel Control Block described in the *Channel Control Block Registers* on page 57. *** TCCB is the Timer Channel Control Block described in the *Timer Channel Control Block* on page 57. (1) Usable on options 100, 200, and 300 only.
(2) Usable on options 200 and 300 only.
(3) Usable on option 300 only.

Board ID/Configuration Buffer

The ID / Configuration buffer (Table 3-2 below) contains the board ID at offset $0000 and the board configuration at offset $0001. The board ID is always $25 for the VMIVME-2540, and the configuration byte is the number of 16-bit channels available on the board. The buffer is updated upon completion of powerup or front panel reset initialization and self-test, and should not by altered by the user. The ID buffer value corresponding to the VMIVME-2540 ordering options is given in Table 3-3 below. During reset initialization or self-test, the board ID location value is undefined.

Table 3-2 Board ID/Configuration (Offset $0000)

Bit 15		Bit 0
16-bit Board ID / Configuration Value		

Table 3-3 VMIVME-2540 ID/Configuration Values

Order Option	Configuration	ID Buffer Value
-000	Four Channels	$2500
-100	Eight Channels	$2501
-200	Sixteen Channels	$2502
-300	Twenty-Four Channels	$2503

Firmware Revision Level

The Firmware Revision Level code is a 4-digit hexadecimal value initialized at completion of self-test. The firmware revision code has *major* and *minor* levels formatted as bytes, that is, $0118 is interpreted as version 1.24. During powerup or front panel reset initialization and self-test, the firmware revision code is undefined.

Table 3-4 Firmware Revision Level (Offset $0002)

Bit 15		Bit 0
Firmware Revision Code		

Command Code

The Command Code is a 16-bit value written to the command code location (offset $0004) of the share memory area by the VMEbus host to instruct the VMIVME-2540 local CPU to perform an operation. A write operation to the command code location results in a command interrupt for the VMIVME-2540 local CPU. The command code may also be written as an 8-bit byte to location $0005. The list of valid command codes are contained in Table 3-5. The command code is initialized to $0000 at powerup, after a reset, or after an *initialize* command. Command descriptions are presented in *Command Status Codes* on page 63.

NOTE: A read operation to the command location from the VMEbus does not assert the command interrupt to the local CPU.

Table 3-5 Command Code (Offset $0004)

Bit 15		Bit 0
16-bit Command Code		

Command Status Code

The command status code is an 8-bit value (Table 3-6 on page 54) placed in the status latch by the VMIVME-2540 local CPU as a response to a command from the host program. The command status code is returned via the shared memory space when command processing has completed. The command status is accessible in the shared memory area at word offset $0006 or byte offset $0007. The command status read operation by the host will occur with minimal DTACK* delay for the access. The host processor VMEbus access to this location on the VMIVME-2540 does not arbitrate for local CPU bus resource. This feature permits continuous polling by the host processor without degrading local CPU performance. However, this feature allows simultaneous access to the latch by both the local CPU and the host processor which may result in bus transition states being read by the host processor via the VMEbus. To reduce problems with transition data, the use of a paced access (using delays between reads) is recommended for the detection of a nonzero command status. After host detection of a nonzero command status, a second read should be performed (with or without delay) and then verified as matching the first. This double read

verification method should be used even if the host processor is providing a paced access to the latch using delays between consecutive reads. If the command status IRQ has been enabled by the host for command status delivery, the local CPU asserts that IRQ and vector after placing the command status code in the latch. This eliminates the read verification requirement, and a command status buffer read will always retrieve stable and valid data. The command status word is initialized to $FF00 after powerup or reset. Refer to the *Command Status Codes* on page 63, and Table 3-21 on page 63.

Table 3-6 Command Status Code (Offset $0006)

Bit 15		Bit 8	Bit 7		Bit 0
Undefined / Not Used			8-bit Status Code		

Command Status Interrupt Request Level

The command status interrupt control byte contains the VMEbus interrupt request level and interrupt enable for command status reporting from the VMIVME-2540 local CPU back to the VMEbus host. This interrupt is used exclusively for command status reporting by the VMIVME-2540, and eliminates the need for host polling to obtain command completion and status information when commands are issued to the VMIVME-2540. All commands written to the board will be followed by a nonzero command status code when processing is completed and includes both reporting of an error-free command status (*command acknowledge*) and error status code. The *clear command status* and the *read...* commands are exceptions. The *clear command status* reports a $00 command status for successful processing of the command, and should always precede a command to initiate any operation of the VMIVME-2540. The *read...* commands (event count and quadrature measurement operations) will report a status of *measurement ready* for successful command processing, or an error status if the *read...* command processing fails. The read command must be issued by the host processor to acquire the discrete (single) measurement data for event count and quadrature operations. The *read* command should only be issued after a channel has been set up for the desired measurement operation by sending either event count or quadrature commands.

The interrupt request level is a 3-bit field using codes $01 through $07 for IRQ1 through IRQ7. The interrupt enable control bit is used by the local CPU to sent the indicated IRQ level if set to *one* with no VMEbus interrupt asserted for a *zero* enable control value. An IRQ level code of zero is defined as no interrupt. The IRQ level code is initialized to $00.

Table 3-7 Command Status Interrupt Request Level (Offset $0008)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08
0	0	0	0	Interrupt Enable	IRQ bit 2	IRQ bit 1	IRQ bit 0

Command Status Interrupt Vector

The command status interrupt vector byte contains the VMEbus interrupt vector placed on the VMEbus during a command status response interrupt acknowledge cycle. The interrupt vector is initialized to $00.

Table 3-8 Command Status Interrupt Vector (Offset $0009)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
User-Defined Interrupt Vector							

Channel ID

The channel ID byte contains a 5-bit field containing the channel number to be associated with a user command. Valid channel ID codes range from $00 for channel 0 through $17 for channel 23; 0 for timer 0 (accesses TCCB 0) through $05 for timer 5 (accesses TCCB 5). The channel ID is initialized to $0000.

Table 3-9 Channel ID (Offset $000A)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08
0	0	0	Channel ID Code				

Continuous/Discrete Flag

The continuous / discrete flag is used by the VMEbus host processor to define the mode to be used for an input measurement operation. The VMEbus host writes either a zero (0) in this location to request that discrete measurement mode of operation (a single measurement by the VMIVME-2540 board, or the value $FF to indicate a continuous measurement is to be performed by the local CPU on the selected channel. A channel configured for continuous measurement (code $FF) results in the local processor reading the channel counter(s) as rapidly as possible, processing the acquired data and placing the resulting measurement data in the channel control block. This flag is cleared by the local CPU after the completion of powerup / self-test / initialization processing, and is read each time a command is sent by the host program. The flag is never written by the local CPU after being cleared at powerup / reset / initialize.

A channel configured for discrete measurement operation will result in a single measurement being placed in the channel control block. For some measurement operations, a *read...* command must be issued by the host processor to acquire the single measurement data (event count and quadrature measurement operations). The *read* command should be issued only after a channel has been set up for the desired measurement operation. The continuous / discrete flag is initialized to $00.

Table 3-10 Continuous/Discrete Flag (Offset $000B)

Bit 7							Bit 0
Continuous / Discrete Flag							

Measurement Ready Flag

The measurement ready flag is a 16-bit parameter which is used by the local CPU to indicate the availability of input measurement data for the host processor. The measurement ready flag is a part of the interface associated with the local CPU measurement queue. It is used in conjunction with the measurement channel ID (location $000E and channel measurement status (location $000F) to manage the data in the measurement queue. The flag is used by the local CPU to indicate to the VMEbus host that a channel ID and channel status data have been moved from the internal queue to the channel ID and status buffers at $000E and $000F. The local CPU then sets the measurement ready flag to $FFFF. When the host completes the reads of the measurement channel and status code, the host should then clear the measurement ready flag (write to zero) in preparation for receiving additional measurement parameters from the local CPU's internal measurement data queue. Unlike the command status register, host access to the measurement flag is arbitrated by the local CPU on a cycle-by-cycle basis. Due to this local bus arbitration, rapid polling of the measurement flag by the VMEbus host will **degrade the performance** of the local CPU. It is suggested that the VMEbus host instead use the timer function, delay loops, or other delay methodology to pace access to the measurement ready flag. The measurement flag is initialized to $00 at powerup / reset / initialize.

Table 3-11 Measurement Ready Flag (Offset $000C)

Bit 15		Bit 0
Measurement Flag		

Measurement Channel ID

The measurement channel ID is a 5-bit binary code corresponding to the channel associated with the measurement code. Valid channel ID codes range from $00 for channel 0 through $17 for channel 23. The measurement channel ID is initialized to $0000. Refer to *Discrete Data Acquisition Mode* on page 91 for a description of the measurement queue interface.

Table 3-12 Measurement Channel ID (Offset $000E)

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 09	Bit 08
0	0	0	Channel ID Code				

Channel Measurement Status

The channel measurement status is an 8-bit value which indicates the status of a measurement function. The measurement status codes are the *measurement ready* and *error* status codes shown in Table 3-21 on page 63. The measurement status is initialized to $00. Refer to *Command Descriptions* on page 70 for a description of the measurement queue interface.

Table 3-13 Channel Measurement Status (Offset $000F)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8-bit Channel Measurement Status							

Channel Control Block Registers

Each channel of the VMIVME-2540 has a Channel Control Block (CCB) of 16 bytes per channel (to accommodate the maximum number of channels available). The channel control block is used by the local CPU to retrieve the user parameters for input and output operation, and to store measurement data for input operations. The contents of each CCB will vary according to the operation to be performed and detailed information is provided in *Modes of Operation* on page 64. The CCB should always be set up with the correct parameters prior to sending a command (a command code write to $0004) operation. This will avoid command status error conditions because of improperly initialized CCB parameters. The CCB also is used by the local CPU to report channel status information to the host processor (measurements ready, error conditions, etc.). This channel status information is a function of the operation being performed by the channel. The Channel Status codes used by the VMIVME-2540 are shown in Table 3-21 on page 63 and described in *Command Status Codes* on page 63.

Table 3-14 Channel Control Block Registers

Variable Name	Offset
Channel Control Blocks (24)	$0010 to $0180

Timer Channel Control Block

The Timer Channel Control Blocks (TCCBs) are used to access the timer functions available to the user. One timer channel is available for each block of four channels, that is, a four-channel board (option 000) has a single time channel, and a 24-channel board (option 300) has six timer channels available. This shared memory area contains space allocation for the maximum of six timer channel control blocks, but the user will only be capable of utilizing the space according to the board option. The timer channels use as input one of the five available internal clock sources; therefore, no front panel inputs or outputs are available for these channels. The outputs are designed to provide only the VMEbus interrupts for a user periodic interrupt function, allowing user selection of the interrupt period. The timer interrupt function is defined in detail in *Channel Output/Waveform Generation Command Codes* on page 93.

NOTE: TCCB 0, Timer Channel 0, is used internally by the local CPU to provide scheduling of interrupts and measurement queue data to the host processor. The host program may utilize this timer, but is restricted to one millisecond repetition rates for its use.

Table 3-15 Timer Channel Control Block Registers

Variable Name	Offset
Timer Channel Control Blocks (6)	$0190 to $01F0

VMIVME-2540 Continuous/Discrete Flag Buffer

The continuous / discrete flags buffer contains a byte-length flag for each of the channels (maximum of 24). This buffer is used by the local CPU to store the measurement mode of each active channel. The contents of this buffer **should not** be modified by the user or erratic channel measurement operation will result. The local CPU stores the continuous / discrete flag for the channel from location $000B when the user sends the command for a selected channel. This flag is used by the local CPU to control the internal operations during its channel-by-channel processing operations.

Table 3-16 VMIVME-2540 Continuous/Discrete Flag Buffer

Variable Name	Offset
Continuous / Discrete Flags	$01F0 to $0207

VMIVME-2540 Measurement Data Valid Flags Buffer

The measurement data validity flags are stored in a buffer of 24 bytes, one byte for each channel. This flag indicates that the measurement data stored in the associated channel's CCB is valid. The flag is cleared ($00) to indicate that the associated channel's measurement data is invalid (a measurement data update is eminently (within 2 µs pending). The host processor may read this flag to determine if the measurement data can be read. See *Programming Strategies for Input Operations* on page 89 for more information.

Table 3-17 VMIVME-2540 Measurement Data Valid Flags Buffer

Variable Name	Offset
Measurement Data Valid Flags	$0208 to $021F

VMIVME-2540 Firmware Release Information

The firmware release information (Table 3-18 below) contains the firmware revision level, release status, and release date as ASCII strings (revision 1.24 and above). This information should be used when contacting VMIC Customer Support for product support.

Table 3-18 VMIVME-2540 Firmware Release Information

Variable Name	Offset
Firmware Rel. Information	$03E0 to $03FF

VMIVME-2540 Daignostic Buffer

If the front panel Fail LED illuminates at any time after initial powerup / reset, the local CPU is generally in a halt state and will no longer respond to host processor commands. The only action to recover from this condition is to activate the front panel reset switch (or activate the VMEbus reset) to reset the board. Refer to *Auxiliary Commands* on page 100 for other uses of the diagnostic buffer.

NOTE: The user should not write to this area: value initialized / used by local CPU.

Table 3-19 VMIVME-2540 Diagnostic Buffer

Variable Name	Offset
Diagnostic_Buffer	$0400 to $3FFF

Command Interface

The VMIVME-2540 processor recognizes 39 different VMEbus host commands as shown in Table 3-20 below. These commands control the operational modes of the VMIVME-2540 listed in *Modes of Operation* on page 64. In addition, commands provide miscellaneous functions for board initialization, limited debug, development, and data collection operations. The table also denotes the applicability of the IRQ / vector and clock / edge select parameters for each of the commands. An entry of X denotes use of the parameters by the local CPU for the associated operation mode. For those entries not denoted by X, the parameters are not used and the associated CCB entry locations should be treated as *Reserved* by the user. The selection of commands other than those presented in Table 3-20, or the use of the reserved commands, will result in the return of a *Request Denied* ($13) error code in the Command Status register.

Table 3-20 VMIVME-2540 Host Commands

Command	Description	IRQ/VEC	Gate/Edge
$00	Disable Channel	X	
$01	16-bit Event Counter	X	
$02	16-bit Event Counter with Gate	X	X
$03	32-bit Event Counter	X	
$04	32-bit Event Counter with Gate	X	X
$05	Reserved		
$06	Read Event Counter	X	
$07	16-bit Frequency Divider		
$08	32-bit Frequency Divider		
$09	Square Wave Generation		
$0A	Pulse Train Generation		
$0B	Frequency Generation		
$0C	Duty Cycle Generation		
$0D	16-bit Period Measurement	X	
$0E	Enhanced 16-bit Period Measurement	X	
$0F	16-bit Frequency Measurement	X	
$10	Enhanced 16-bit Frequency Measurement	X	
$11	16-bit Pulse-Width Measurement	X	
$12	Enhanced 16-bit Pulse-Width Measurement	X	
$13	Quadrature Position Measurement	X	
$14	Reserved		
$15	Read Quadrature Position	X	
$16	Integer Quadrature Position Measurement	X	
$17	Read Integer Quadrature Position	X	
$18	16-bit Timer	X	
$19	16-bit Timer with Restart	X	
$1A	Disable Timer	X	

Table 3-20 VMIVME-2540 Host Commands (Continued)

Command	Description	IRQ/VEC	Gate/Edge
$1B	Initialize		
$1C	Reset Command Status	X	
$1D	Block Move	X	
$1E	Jump To (Execute)		
$1F	Echo PC		
$20	Integer Period Measurement	X	
$21	Pulse Sequence Generation	X	
$22	Programmed Output		
$23	Quadrature Position Control	X	
$24	Group Acquisition - Integer QPM	X	
$25	Event-Triggered VMEbus Timer	X	X (edge)
$26	16-bit Integer Pulse-Width Measurement	X	X
$27	32-bit Integer Period Measurement	X	X
$28	32-bit Integer Pulse-Width Measurement	X	X

Programming Using the Command Interface

The following steps are required for host programming using the VMIVME-2540 command interface. These steps will ensure proper synchronization and operation of the board.

Initialize the parameters for the selected channel in the channel CCB.

For measurement commands, select the desired measurement mode by writing a $00 for discrete measurement (or $FF for continuous mode) to offset $000B. This parameter is used only for input operations.

Select the channel to be used by writing the channel number to the channel ID at offset $000A.

Send the *clear status* command code (write $001C to offset $0004).

Verify that the command status is cleared by reading $00 at offset $0007. See the description of the command status code for proper access of the command status data.

Send the command for the desired operation (that is, for an event count measurement, write $0001 to offset $0004).

If polling is used, verify that the command processing is completed by reading the command status until a nonzero command status is retrieved (using the access methodology previously described). If the command status interrupt is used, wait for the command status interrupt, and perform a single read to the command status word (or byte).

Process the command status code to detect any error that may have occurred during command processing by the VMIVME-2540 CPU. A *command acknowledge* command status code will indicate a successful command operation. There are exceptions to this command status code for error-free processing. See *Command Descriptions* on page 70, for detailed information. After the host processing is completed for output operations,

output signals will be active with the signal selected by the command code and CCB parameter list.

For measurement operations, the counter(s) for the selected channel will be active with the measurement selected by the command and CCB parameter list for that channel after the command acknowledge status code is received by the host program. For the *read*... commands, the command status code will return a ...*ready* command status if there are no errors during command processing.

Command Status Codes

The VMIVME-2540 local CPU responds to VMEbus commands with status codes relevant to the commanded action as shown in Table 3-21 below. These status codes are described in *Modes of Operation* on page 64 with the corresponding mode of operation. These codes are applicable for both command status and channel status conditions. However, some of the commands, such as *command acknowledge* are applicable only to command write operations by the host. Other commands are applicable only for channel status reporting. Command and channel columns of Table 3-21 indicate the applicability of the status codes. A nonzero command status code is always returned in the command status byte ($0007) in response to a host write operation to the command code register ($0004) with the exception of the clear command status command which returns a zero.

The command status code provides the primary synchronization method between the host program and the VMIVME-2540 by always providing a nonzero status code when the command processing is completed for the command sent by the host program. Failure to wait for a command status word for each command sent can result in fatal errors occurring during command processing by the VMIVME-2540 local CPU.

Table 3-21 VMIVME-2540 Status Codes

Code	Status Description	Command	Channel
$00	NULL	Yes	Yes
$01	Command Acknowledge	Yes	No
$02	Event Count Ready	Yes	Yes
$03	Period Measurement Ready	No	Yes
$04	Frequency Measurement Ready	No	Yes
$05	Pulse-width measurement Ready	No	Yes
$06	Quadrature Position Measurement Ready	Yes	Yes
$07	Limit Alarm	No	Yes
$08	Timer Alarm	No	Yes
$09	Channel Allocation Error	Yes	No
$0A	Bounds Error	Yes	No
$0B	Period Error	Yes	No
$0C	Pulse width Error	Yes	No
$0D	Frequency Error	Yes	No
$0E	Scale Error	Yes	Yes
$0F	Reserved	No	No
$10	Gate Error	Yes	No
$11	Limit Error	Yes	No
$12	Active Channel Error	Yes	No
$13	Request Denied	Yes	No
$14	Under-Range**	No	Yes

NOTE: ** Returned as a status code with a valid measurement. Indicates a higher resolution is obtainable and ranging is active.

Modes of Operation

The VMIVME-2540 supports several modes of input, output, and timing operations. Input operations perform measurements of the user's bi-level input signals. The output mode of operations provide the bi-level, time-based signal generation capabilities of the board, bi-level, time-based output signals as a function of user's input signal, and the timing operations provide periodic time-based events for the host processor.

Each channel's mode of operation is controlled via the entry of parameters into the channel's control block and the writing of the command to invoke the selected mode of operation.

The signal generation and measurement modes may require either a clock input signal, a gate input signal, and / or output connections to the associated channel(s) on the front panel connectors (P3 and P4). Those operations that utilize these signal (external clock, gate, and output) connections are shown in Table 3-22 below. Operations which do not require any external connection (timer operations) are not shown. For some signal generation operations, an output connection is not required for the channel to operate in the selected mode; the channel output is assumed to be connected to either a channel on the board (clock or gate), to the user's field equipment, or not connected. See *Command Descriptions* on page 70 for the connection requirements for the operational modes.

Table 3-22 Front Panel External Clock, Gate, and Output Connections

Command	Clock	Gate	Output
16-bit Event Counter	X		
16-bit Event Counter with Gate	X	X	
32-bit Event Counter	X		
32-bit Event Counter with Gate	X	X	
16-bit Frequency Divider	X		X
32-bit Frequency Divider	X		X
Square Wave Generation			X
Pulse Train Generation			X
Frequency Generation			X
Duty Cycle Generation			X
16-bit Period Measurement		X	
Enhanced 16-bit Period Measurement		X	
16-bit Frequency Measurement		X	
Enhanced 16-bit Frequency Measurement		X	
16-bit Pulse-Width Measurement		X	
Enhanced 16-bit Pulse-Width Measurement		X	
Quadrature Position Measurement	X		
Integer Quadrature Position Measurement	X		
16-bit Integer Period Measurement		X	
16-bit Integer Pulse-Width Measurement		X	
Pulse Sequence Mode		X	X
Programmed Output			X
Quadrature Position Control		X	X
Group Acquisition	X		

Table 3-22 Front Panel External Clock, Gate, and Output Connections (Continued)

Command	Clock	Gate	Output
Delayed Event Timer		X	
32-bit Integer Period Measurement		X	
32-bit Integer Pulse-Width Measurement		X	

Some input and output operations require multiple channels. All 32-bit input operations and the enhanced period, pulse width, and frequency measurement operations require two channels. The quadrature position control operation requires four channels. All multiple channel operations require that contiguous channels be used and that the lower channel of the channel pair is even. The four-channel QPC operation requires four contiguous channels and the lower channel must be on a modulo 4 boundary; that is, 0, 4, 8, 16, etc. The lower channel is always used as the channel ID for issuing commands which utilize multiple channels.

Input Modes of Operation

a. Integer Event Counter with Programmable Limit Count:

1. 16-bit unsigned integer event counter, no level gating, automatic restart.
2. 16-bit unsigned integer event counter, level gating, automatic restart.
3. 32-bit unsigned integer event counter, no level gating, halt at limit; requires two counters.
4. 32-bit unsigned integer event counter, level gating, halt at limit; no interim count available; requires two channels.

b. Floating-Point Period/Frequency Measurement: Internal time base: resolution ranges {0.2 ms, 2 ms, 20 ms, 200 ms, 2 ms}.

1. 16-bit counter with N-sample averaging; from 400 ns minimum period with 200 ns resolution to 131 s maximum period with 2 ms resolution.
2. 16-bit counter with 16-bit prescaler and N-sample averaging requires two counters, 858.9 second maximum period with 0.2 µs resolution; requires two contiguous channels.

c. Floating-Point Pulse-Width Measurement:

1. 16-bit counter with N-sample averaging and autoranging; from 400 ns minimum pulse width with 200 ns resolution to 131 s maximum pulse width with 2 ms resolution.
2. 16-bit counter with 16-bit prescaler and N-sample averaging requires two counters, 858.9 second maximum pulse width with 0.2 µs resolution; requires two contiguous channels.

d. Floating-Point Quadrature Position Measurement: User-specified floating-point scale factor, 1/4 wave resolution, 32-bit counter with user-specified CW/CCW limits.

1. Sin/Cos Quadrature inputs; requires two contiguous channels.

e. **Integer Quadrature Position Measurement:** 1/4 wave resolution, 32-bit two's complement data format, overflow alarm interrupt, requires two contiguous channels.

f. **16-bit Integer Period Measurement:** 16-bit unsigned counter with internal time base, autoranging, or specified clock source.

g. **Group Acquisition:** Acquires two integer QPM measurements simultaneously, requires four contiguous channels.

h. **16-bit Integer Pulse-Width Measurement:** 16-bit unsigned counter with internal time base, autoranging, or user-selected clock period.

i. **32-bit Integer Period Measurement:** 32-bit unsigned counter with user-selected clock period, requiring two contiguous channels.

j. **32-bit Integer Pulse-Width Measurement:** 32-bit unsigned counter with user-selected clock period, requiring two contiguous channels.

k. **Time-Delayed Event with VMEbus Interrupt:** Generates a VMEbus interrupt on receipt of an edge input. The interrupt is delayed by a user-selectable time of 200 to 13.2 ms. It is specified in 16-bit floating-point. Repetitive operation is available using the automatic restart flag.

Output Modes of Operation

a. **16-bit Frequency Divider:**

1. Generates an output signal as a function of an input signal.
2. User-selected 16-bit unsigned integer frequency divisor between 2 and 65,535.

b. **32-bit Frequency Divider:**

1. Generates an output signal as a function of an input signal.
2. User-selected 32-bit unsigned integer frequency divisor between 2 and 4,294,967,295, external supplied gate; requires two contiguous channels.

c. **Period Pulse-Width Generation Frequency/Duty Cycle Generation:** 16-bit counter with internal time base: 1 ms minimum/131.1 s maximum period.

d. **Pulse Sequence Mode:** 16-bit integer specification of period, pulse width, and number of pulses; interrupt upon completion; requires two contiguous channels.

e. **Programmed Output:** Simple digital output capability.

f. **Quadrature Position Control:** Generation of sequence of pulses in quadrature; user specifies clock source, 16-bit integer pulse width, and 16-bit two's complement quadrature position delta; requires four contiguous channels.

Timing Modes of Operation

a. **Timer/Periodic Interrupt:** Internal time base, 1 ms minimum/131.1 s maximum period.

1. 16-bit timer: Interrupts VMEbus host after terminal count.

b. **16-bit Periodic Interrupt:** Same as 16-bit timer with automatic restart.

Channel Control Blocks Common Parameters

The channel control block structure for the VMIVME-2540 is shown in Table 3-23 below with locations shown for some of the parametric entries for the VMEbus IRQ / vector codes, edge / gate codes, and clock period select codes which are common for some, but not all, operations supported by the VMIVME-2540. (The parameter's locations may also differ for some operations requiring these entries in the CCB.) For those operations which use these common parameters (denoted by X in Table 4.3-1), the format is identical for the channel control block entries and the codes entered. The position of the clock period select and gate / edge codes may vary for some operations, primarily for operations which require both parameters. For the applicable operations, the selected channel CCB is initialized with the desired VMEbus IRQ / vector codes, gate / edge codes, clock period selects, and all other parameters applicable before issuing a channel command by writing to the command code. **Reserved locations in the CCBs should not be modified by the user or else the operation of the corresponding channel will be unpredictable.** The information contained in the **Reserved** locations should not be used for host program processing flow control (Exception: The first location of the CCB will always contain the command code after successful completion of the commanded operation. This location may be **read** by the host program to determine an active / inactive channel status. This methodology **should not be used** in lieu of channel status verification for verifying successful command processing by the VMIVME-2540.) Observing these rules will ensure host software compatibility with future enhancements or upgrades to the VMIVME-2540 product. The gate / edge code has four possible values for user-selection of active gate and clock input signals, as shown in Table 3-24 on page 68. The VMEbus IRQ / vector byte locations have the same format as the command status IRQ and command status vector bytes described earlier. The VMEbus IRQ for a channel control block may be enabled by setting bit 3 of the IRQ location to *one* and disabled by clearing bit 3, or simply filling the IRQ location with $00.

The clock period select codes control the time base for some, but not all, of the VMIVME-2540 input and output operations. The available clock periods supported by the board and their associated codes are shown in Table 3-25 on page 68.

Table 3-23 Typical CCB Format/Common Parameters

Offset	Function
$00	Reserved
$01	Clock Period Select or Gate / Edge Code
$02	VMEbus IRQ
$02	VMEbus Vector
$04 to $0F	Parameter List / Channel Status

Table 3-24 Gate/Edge Codes

Offset	Function
$00	Active-High Gate, Rising Clock Edge
$01	Active-High Gate, Falling Clock Edge
$02	Active Low Gate, Rising Clock Edge
$03	Active-Low Gate, Falling Clock Edge

Table 3-25 Clock Period (Time Base) Select Code

Scale Code	Time Base
$00	Autoranging
$01	200 ns Per Count
$02	2 µs Per Count
$03	20 µs Per Count
$04	200 µs Per Count
$05	2 ms Per Count

Operation Mode Selection Flag

The following input functions permit the user to select the mode of operation for the function:

- Floating-point period, pulse width, and frequency measurement
- Enhanced floating-point period, pulse width, and frequency measurement
- 16-bit integer period and pulse-width measurement
- 32-bit integer period and pulse-width measurement

The CCBs for these operations use an *Operational Mode Select Flag*. Normally when a counter receives a triggering input signal (a rising or falling edge for period and frequency measurement or a rising and following edge for pulse-width measurement) via the front panel, the counter/timer hardware will eventually overflow if this input is not followed by a second signal. This will generate a terminal count local interrupt and disarm the counter from further counting. This condition is normally an error condition, possibly indicating a loss of signal. A scale error code ($0E) is returned when the counter overflow occurs. The operational mode select can be used to notify the local CPU to rearm the counter, if desired. This will result in the counter being rearmed and counting will continue. Otherwise, the user must disable the associated channel and re-enable the channel by sending the appropriate command. The scale error code will be returned for the overflow condition, regardless of the rearm mode of operation, and must be acknowledged by the host computer (by clearing the code) before further *measurement ready* status codes will be reported. However, measurements will continue to be updated to the CCB for the channel.

Format of the Operation Mode Select Flag:

Operational Mode Select Flag

Table 3-26 Operational Mode Select Flag

Bit 07	Bit 06	Bit 05	Bit 04	Bit 03	Bit 02	Bit 01	Bit 00
Reserved (0)							Auto. Rearm**

Operational Mode Flag Bit Descriptions

Bits 07 through 01: **Reserved** - Reserved bit are set to zero (0).

Bit 00: **Auto Rearm** - A zero (0) in this bit location = NO, a one in this bit location = YES. To set the Automatic Rearm , set the operational mode select flag to $01.

Command Descriptions

The following sections describe the command codes and the usage of the command codes for performing the operations supported by the VMIVME-2540 module.

The required front panel connections are also described for each of the commands that require an input signal, or the use of an output signal connection for proper channel operation.

Initialization and Synchronization Command Codes

A portion of the command codes provide for the initialization of the VMIVME-2540 and individual channels. An additional command provides the host program with the capability to initialize the command status prior to writing the command. Command status interrogation provides the primary synchronization mechanism with the VMIVME-2540 for the host program.

Command $00: Disable Channel. The disable channel command instructs the VMIVME-2540 local CPU to disarm the counter(s) associated with the currently active command for the selected channel. The CCB(s) contents for the selected channel is cleared (set to zeros). This command must be issued by the user for any active channel before commanding any mode of operation. If the user commands a mode of operation for an already active channel, the VMIVME-2540 local CPU returns the *active channel error* command status code. The disable channel command may be issued to any channel configured for 16-bit operation. A channel pair configured for 32-bit operation must be disabled by issuing the disable channel command to the lower (even) channel number. The QPC operation must be disabled by issuing the disable channel command to the lowest (even) channel of the four-channel group. If the user issues the disable channel command for other than the correct channel for an active multichannel operation, the VMIVME-2540 local CPU reports the *channel allocation error* status code. A *disable channel* command can be sent for an inactive (disabled) channel, and a normal *command acknowledge* command status will be returned to the host processor.

Command $1B: Initialize. This command instructs the VMIVME-2540 local CPU to initialize all shared memory variables and AM9513A system timing controllers to the same state as that which immediately follows self-test from powerup or front panel reset. All channels are disabled, channel outputs are set to zero, and all channel CCBs are cleared. The *initialize* command requires only milliseconds to execute and is appropriate for execution when a host program restart is initiated.

Command $1C: Reset Command Status Buffer. This command instructs the VMIVME-2540 local CPU to write $xx00 to the command status buffer, located at offset $0006 from the base address. As noted in the command status buffer description, the upper byte of the command status buffer is undefined, while the lower byte is cleared by this command. The response time for this command is less than 50 µs. This command provides the necessary synchronization with the VMIVME-2540 local processor when polling is used to transfer commands. Proper command-to-command status sequencing requires that the *reset command status* command be issued first, then the command status buffer be verified for a zero value. The desired operational command is then issued to the VMIVME-2540, and a command status is verified by reading a nonzero status, followed by a second read which is verified to match the previous nonzero read. This loop should be repeated

until matching nonzero data is detected on consecutive accesses, and generally the reread match will occur on the second attempt. This method will avoid transition data problems due to the nonarbitrated operation of the command status buffer.

Channel Input/Measurement Command Codes

The following sections describe the input operations of the VMIVME-2540 counter timer board. The operations were designed to function strictly for the acquisition of data via the front panel. However, the counters configured to support these operations will generally have an associated pulse output signal at the P3/P4 output pins. This signal may or may not have a useful function for the user's application.

Integer 16-bit Event Counting

The 16-bit event counter CCB, shown in Table 3-27 below, requires the host to initialize the gate/edge code, IRQ/Vector, and limit count parameters prior to writing the event counter command code for the selected mode of operation. All signal rising (or falling) edges will be counted for this command. The local CPU configures the counter of the selected channel for counting. Counting then proceeds independent of the local CPU after counting is enabled, and the count is accumulated in the internal counter of the AM9513A. The internal counter value is read via the local CPU using either the continuous measurement mode (*Continuous Data Acquisition Mode* on page 90) or the *read event count* command (*Integer 32-bit Event Counting* on page 72, Command $06). If the event count reaches the limit count value and the VMEbus IRQ is enabled, the VMIVME-2540 local CPU asserts the IRQ level specified by the CCB. If the VMEbus IRQ is not enabled and the limit count value is reached, the VMIVME-2540 local CPU posts the *limit alarm* code to the local message queue and to the CCB status byte. The 16-bit event counter modes automatically restart counting when the limit count is reached. Exception: Event counting with a limit count of one will not restart; only a single VMEbus interrupt will be generated, if enabled. Use Event-Triggered VMEbus timer with minimum time delay instead for single-event with automatic restart.

NOTE: Event count measurements are available only via the *read event count* command (*Integer 32-bit Event Counting* on page 72) or the continuous measurement mode. The measurement queue and the interrupt events are used only for limit status reporting.

Table 3-27 16-bit Event Counter Channel Control Block

Offset	Function
$00	Reserved
$01	Gate/Edge Code
$02	VMEbus IRQ
$03	VMEbus Vector
$04	16-bit Limit Count
$06	16-bit Current Count
$08 to $0B	Reserved
$0C	Channel Status
$0D to $0F	Reserved

Command $01: 16-bit Event Counter, No Level Gating. The VMIVME-2540 local CPU configures the indicated channel for event counting with indicated clock edge and no level gating. The counter will count up until the limit count is reached at which time the limit alarm will be placed in the measurement queue and the CCB channel status byte. The counter will then restart event counting from zero. The continuous measurement mode is available for this command and will return the *event count ready* status to the CCB channel status byte as event counts are placed in the *current count* word location of the CCB.

NOTE: If the event counter *limit alarm* status is placed in the CCB channel status byte, the user must acknowledge the alarm condition (by clearing the status byte) before any subsequent *event count ready* status will be reported in the CCB channel status byte. The current count will continue to be updated in the continuous measurement mode if the limit alarm is not acknowledged by the host processor.

Command $02: 16-bit Event Counter, Level Gating. The VMIVME-2540 local CPU configures the indicated channel for event counting with indicated clock edge and level gating. Continuous measurement mode is available for this command (see above).

Front Panel Input Signal Connections: The event count signal is connected to the clock input of the desired channel with the event gate signal (command $02 only) attached to the corresponding gate input.

Integer 32-bit Event Counting

The parameter list for the 32-bit event counter CCB contains a 32-bit unsigned integer limit count supplied by the user prior to the command interrupt, and a 32-bit unsigned integer current count value returned by the VMIVME-2540 local CPU in response to a Read Counter command. This CCB format is shown in Table 3-28 on page 73. The 32-bit event counter command may be issued for even channels only: channels 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, and 22. If the user commands a 32-bit event counter mode for an odd channel number, the VMIVME-2540 local CPU gives the *channel allocation error* command status code.

For a channel pair to be successfully set up for the 32-bit counting modes, both the even and next higher odd channels must first be inactive (disabled). If the user commands a 32-bit event counter mode for a channel pair which has not previously been disabled, the VMIVME-2540 local CPU gives the *channel allocation error* command status code.

The 32-bit limit count must be a quantity greater than 65,536. If the 32-bit limit count is 65,536 or less, the VMIVME-2540 local CPU gives the *bounds error* status code. If the event count reaches the limit count value and the VMEbus IRQ is enabled, the VMIVME-2540 local CPU asserts the IRQ level specified by the CCB. If the VMEbus IRQ is not enabled and the limit count value is reached, the VMIVME-2540 local CPU posts the *limit alarm* code to the local message queue and to the channel status byte. The 32-bit event count operation does not automatically restart upon reaching limit count value.

Table 3-28 32-bit Event Counter Channel Control Block

Offset	Function
$00	Reserved
$01	Gate/Edge Code
$02	VMEbus IRQ
$03	VMEbus Vector
$04	32-bit Limit Count
$08	32-bit Current Count
$0C	Channel Status
$0D to $0F	Reserved

Command $03: 32-bit Event Counter, No Level Gating. The VMIVME-2540 local CPU configures the indicated channel pair for 32-bit event counting with indicated clock edge and no level gating. This mode of operation counts events until the limit count is reached, after which the counter pair is disabled. Continuous measurement mode is available for this command with *event count ready* status reported in the CCB status byte as measurements are updated by the local CPU.

NOTE: The *limit alarm* channel CCB status will override subsequent *event count ready* status code reporting until the alarm status is acknowledged by the host program (by clearing the *limit alarm* status code in the CCB channel status).

Command $04: 32-bit Event Counter, Level Gating. The VMIVME-2540 local CPU configures the indicated channel pair for 32-bit event counting with indicated clock edge and level gating. This mode of operation counts events while the gate is active (as selected by the gate/edge code) and disregards events when the gate is inactive. Counting will continue until the limit count is reached, after which the counter pair is disabled. Continuous measurement mode is available for this command. However, the internal counter contents cannot be read when a gating signal is used, hence the event count will always contain zero, and the user will only be able to detect a limit alarm via the CCB channel status.

Command $05: Reserved.

Command $06: Read Event Count. For channels configured by commands $01 through $04 only, the current count of the selected channel is read and returned into the current count variable of the CCB. The command status buffer and the CCB channel status is loaded with the *event count ready* command status code and the VMEbus IRQ and vector are placed in the VMEbus interrupt queue if nonzero. The 32-bit event counter with gating enabled does not allow the current count value to be read due to the limitations of the AM9513A system timing controller modules. Minimum data transport lag is approximately 100 µs.

Front Panel Input Signal Connections: The event count signal is connected to the clock input of the lower (even) channel and the (optional) event gate signal is connected to that channel's gate input.

Period Measurement

This CCB format is shown in Table 3-29 on page 75. The parameter list for the period measurement CCB contains a clock select code, VMEbus IRQ and interrupt vector, average value, and a 32-bit IEEE-754 floating-point period measurement value. The clock select is used to select the clock period to be used during 16-bit period measurement, as shown in Table 3-25 on page 68.

If the clock select code contains any value other than those listed in the above table, the VMIVME-2540 local CPU returns the *scale error* command status code. The autoranging mode will result in the local CPU varying the time base to obtain the best resolution for the period measured. The autoranging process is repeated for each new measurement and is initiated with the highest resolution clock selection (200 ns.). The use of autoranging will increase the data transport lags for the measurements accordingly. If the period of the input signal results in a counter overflow for the selected clock (or the 2 ms clock for autoranging), the VMIVME-2540 local CPU places the *scale error* code in the CCB channel status at offset $0C, and will queue the VMEbus IRQ for delivery, if enabled. For discrete measurements, if a period measurement error occurs, *scale error* and corresponding channel ID are placed in the local measurement queue, if the VMEbus IRQ is disabled. The clock select code is not used for enhanced 16-bit period measurement. The sample size for the average value, N, has the range of values 0 to 65,535 ($0000 to $FFFF). Sample size zero (0) has the same effect as one (1), measuring the signal period only once.

The enhanced 16-bit period measurement mode requires two channels and can only be commanded for even channels: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, and 22. If the user commands enhanced 16-bit period measurement for an odd channel number, the VMIVME-2540 local CPU gives the *channel allocation error* response code. Both the even channel and the next higher odd channel must be inactive (disabled) for a channel pair to be successfully set up for enhanced 16-bit period measurement. If the user commands enhanced 16-bit period measurement for a channel pair and either channel is active, the VMIVME-2540 local CPU gives the *channel allocation error* response code.

The time base for enhanced 16-bit period measurement is the internal 5 MHz STC clock. The lower (even) counter of the channel pair is a 16-bit prescaler which clocks the upper (odd) counter of the channel pair, permitting the user to establish the clock period for the period measurement (autoranging is not used for the enhanced mode of operation since the user controls the clock period via the prescaler selection). The user may specify prescaler values ranging from 2 through 65,535. This will yield a 400 ns resolution for periods up to 26.2 ms using a prescalar of 2 and 13.1 ms resolution for periods up to 858.97 s using a prescalar value of 65,535.

Table 3-29 Period Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Clock Select Code ($00 to $05)
$02	VMEbus IRQ
$03	VMEbus Vector
$04	Sample Size for Average Value (16-bit Integer)
$06	Reserved
$08 to $0B	32-bit IEEE-754 Floating-Point Period Measurement (seconds)
$0C	Channel Status
$0D	Operational Mode Select Flag (page 69)
$0E	Enhanced 16-bit Prescaler
$0F	Reserved

Command $0D: 16-bit period measurement, N-sample size average. After N-measurements, the VMIVME-2540 local CPU converts the accumulated value into IEEE-754 format, divides by N, places the quotient in the period value location, places the *period measurement ready* status in the CCB and queues the host VMEbus interrupt for delivery (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The input signal is connected to the gate input of the selected channel.

Command $0E: 16-bit enhanced period measurement, N-sample size average. The user specifies the number of 5 MHz clock cycles in the enhanced 16-bit prescalar location of the CCB, writes the IRQ / vector, the sample size, and clears the channel status prior to writing the code to the command word (or byte). After N-measurements, the VMIVME-2540 local CPU converts the accumulated value into IEEE-754 format, divides by N, places the quotient in the period value location, places the *period measurement ready* status in the CCB, and queues the host VMEbus interrupt for delivery (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The input signal is connected to the gate input of the upper (odd) channel of the channel pair.

Frequency Measurement

The CCB format for this measurement is shown in Table 3-30 on page 76. The parameter list for the frequency measurement CCB contains a scale code, VMEbus IRQ and interrupt vector, average value, and a 32-bit IEEE-754 floating-point frequency value. The scale code for 16-bit frequency measurement may take on the values specified in Table 3-25 on page 68 with the same restrictions described for period measurement. If there is a time base-related error in initiating frequency measurement, the *frequency error* status code is returned for the command status. The sample size parameter N has the range of values 0 through 65,535 ($0000 through $FFFF). Sample size 0 has the same effect as 1, measuring the signal period only once.

The enhanced 16-bit frequency measurement mode allows the user to select the clock period for the measurement, ranging from 400 ns to 131.1 ms, using prescalar codes between 2 and 65,535. Enhanced frequency measurements require two channels and can only be commanded for even channels: 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, and 22. Commanding enhanced 16-bit frequency measurement for an odd channel will result in a *channel allocation error* for the command status.

Table 3-30 Frequency Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Time Base Scale Code
$02	VMEbus IRQ
$03	VMEbus Vector
$04	Sample size for average value (16-bit integer)
$06	Reserved
$08 to $0B	32-bit IEEE-754 Floating-Point Frequency Measurement (Hz)
$0C	Channel Status
$0D	Operational Mode Select Flag (page 69)
$0E	Enhanced 16-bit Prescaler
$0F	Reserved

Command $0F: 16-bit frequency measurement, N-sample size average. After N-measurements, the VMIVME-2540 local CPU converts the accumulated value into IEEE-754 format, divides N by the sum, places the quotient in the frequency result location, places the *frequency measurement ready* status in the CCB and queues the host VMEbus interrupt for delivery (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The signal to be measured is connected to the gate input of the selected channel.

Command $10: Enhanced 16-bit frequency measurement, N-sample average. After N-measurements, the VMIVME-2540 local CPU converts the accumulated value into IEEE-754 format, divides N by the sum, places the quotient in the frequency result location, places the *frequency measurement ready* status in the CCB, and queues the host VMEbus interrupt (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The signal to be measured is connected to the gate input of the upper (odd) counter of the channel pair.

Pulse-Width Measurement

This CCB format is shown in Table 3-31 on page 77. The parameter list for the pulse-width measurement CCB contains a scale code, VMEbus IRQ and interrupt vector, sample size value, and a 32-bit IEEE-754 floating-point pulse width value. The time base scale code for 16-bit pulse-width measurement can take on the values specified in Table 3-25 on page 68 with the same restrictions described for the period

measurement. If there is a time base-related error in measuring the pulse width, the *pulse width error* code is returned. The sample size value N has the range of unsigned integer values 0 through 65,535 ($0000 to $FFFF). Sample size 0 has the same effect as 1, measuring the signal period only once.

The enhanced 16-bit pulse-width measurement mode may be commanded only for even channels: 0, 2, 4, 6, 8,10, 12, 14, 16, 18, 20, and 22. The prescalar parameter has the same attributes and effect as that for period and frequency measurements. Commanding enhanced 16-bit pulse-width measurement for an odd channel will result in a command status code of *channel allocation error.*

Table 3-31 Pulse-Width Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Time Base Scale Code
$02	VMEbus IRQ
$03	VMEbus Vector
$04	Sample Size Value (16-bit Integer)
$06	Reserved
$08 to $0B	32-bit IEEE-754 Floating-Point Pulse-Width Measurement (seconds)
$0C	Channel Status
$0D	Operational Mode Select Flag (page 69)
$0E	Enhanced 16-bit Prescaler
$0F	Reserved

Command $11: 16-bit Pulse-Width Measurement, N-sample size average. After collecting N-samples, the VMIVME-2540 local CPU transforms the accumulated value into IEEE-754 format, divides by N, places the quotient in the period result location, places the *pulse width ready* status in the CCB and queues the host VMEbus interrupt for delivery (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The signal to be measured is connected to the gate input of the selected channel.

Command $12: Enhanced 16-bit Pulse-Width Measurement, N-sample average. After collecting N-samples, the VMIVME-2540 local CPU converts the accumulated value into IEEE-754 format, divides by N, places the quotient in the period result location, places the *pulse width ready* status in the CCB, and queues the host VMEbus interrupt for delivery (if enabled). Continuous measurement mode is available for this command.

Front Panel Input Signal Connections: The signal to be measured is connected to the gate input of both counters of the channel pair.

Quadrature Position Measurement

The CCB format for quadrature position measurement is shown in Table 3-32 on page 79. Quadrature position measurement requires two consecutive channels for operation with a clockwise input signal on the even channel and a counterclockwise input signal on the odd channel. The direction sense of the measurement is determined by the quadrature relationship of the input signals. When the even channel leads the odd channel, the direction is defined to be *clockwise*. For nonrotational position measurement, this direction may alternately be defined as *forward*, *up*, *left*, or *right* as dictated by the linear application. In the following discussion of angular scale factors, clockwise and counterclockwise limit angles, the units of measurement apply equally to linear motion by substituting units of distance. The QPM channel control block is shown in Table 3-32 on page 79.

The VMIVME-2540 has front-end logic for processing quadrature position clock inputs. Each edge of the clock inputs creates a clock pulse which is steered by the direction indicator. Clockwise motion creates four clockwise pulses for each period of the clock inputs and no counterclockwise pulses. Similarly, counterclockwise motion creates four counterclockwise pulses for each period of the clock inputs and no clockwise pulses. The effect of the STC front-end logic is to multiply by four the resolution of the external quadrature encoder. Since all inputs to the VMIVME-2540 are synchronized to the 5 MHz time base, the maximum frequency of the quadrature input signals must be less than 1.25 MHz for reliable operation.

Two 16-bit counters in the AM9513A STC record the clockwise and counterclockwise pulses generated by the front-end logic. These 16-bit hardware counters are augmented by 16-bit *software counters* using variables in local CPU memory. Absolute position is calculated by subtracting the counterclockwise 32-bit count from the clockwise 32-bit count and converting the result to floating-point format using the quadrature scale factor. Limit checking is performed on the 32-bit difference count using values derived from the user-supplied clockwise and counterclockwise limits. When a limit is exceeded, the VMIVME-2540 local CPU writes the corresponding limit code to CCB channel status at offset $01 and asserts the VMEbus interrupt from the CCB, if enabled. If the VMEbus interrupt IRQ is not enabled, the *limit alarm* status code is posted to the message queue with the corresponding quadrature channel number. Table 3-33 on page 79 shows the QPM limit and overflow codes which may be returned to the user in the CCB channel status.

Since the quadrature position is not recorded with an *up-down* counter, the possibility exists that jitter at the quadrature encoder shaft will cause repetitive direction changes and succeeding trains of clockwise and counterclockwise clock pulses. In this case, each 32-bit count variable would be incremented until overflow occurred. The VMIVME-2540 local CPU detects this case and normalizes the two 32-bit count variables by subtracting the smaller count from the larger count, then clearing the smaller count. A 32-bit quadrature position overflow can also occur as a result of continuous rotation (or translation) in either direction. When the overflow condition occurs, the VMIVME-2540 local CPU writes the corresponding overflow code to CCB channel status at offset $01 and queues the VMEbus interrupt specified in the CCB, if enabled. If the VMEbus interrupt IRQ is not enabled, the *limit alarm* message is posted to the message queue with the corresponding quadrature channel number.

Table 3-32 QPM Channel Control Block

Offset	Function
$00	Reserved
$01	Channel Status
$02	VMEbus IRQ
$03	VMEbus Vector
$04	32-bit Floating-Point Scale Factor
$08	32-bit Clockwise Limit Angle
$0C	32-bit Counterclockwise Limit Angle
$10	Reserved
$11	Direction Indicator
$12 to $13	Reserved
$14	32-bit Floating-Point Current Angle
$18 to $1F	Reserved

Table 3-33 QPM Channel Status Codes

Code	Description
$01	Clockwise 32-bit Overflow
$02	Counterclockwise 32-bit Overflow
$03	Clockwise Limit Exceeded
$04	Counterclockwise Limit Exceeded

Command $13: Quadrature Position Measurement. The user initiates quadrature position measurement for a given channel pair by first writing the 32-bit floating-point scale factor, clockwise limit angle, and counterclockwise limit angle values to the lower (even) channel control block. For example, an encoder which outputs 7200 pulse / revolution a scale factor of 0.05 would be used (360 degrees / 7200 pulses). If a limit alarm was desired at 225 degrees, regardless of the rotational direction then 225.0 and 135.0 degrees would be entered for the clockwise and counter clockwise limit values, respectively.

If a host processor interrupt is desired whenever a limit is exceeded or 32-bit overflow occurs, the VMEbus IRQ and vector are also initialized in the channel control block. The user next writes the lower (even) channel ID code to the channel ID variable and issues the QPM command to the VMIVME-2540 local CPU. When the *command acknowledge* command status code is returned, quadrature position measurement is active for the indicated channel pair at absolute position zero. Continuous

measurement mode is available for this command. Any *limit alarm/overflow* status must be acknowledged by clearing the alarm status in the channel status byte or further *QPM ready* status codes will not be reported. Measurements will continue to be made and reported under the limit / overflow condition with user responsibility for maintaining total position information after an overflow condition has occurred. Continuous measurement mode eliminates the need for issuing the read quadrature position command to obtained updated position data.

NOTE: QPM current angle data is available only via the *Read QPM* command or the continuous measurement mode. The measurement queue and the interrupt events are used only for limit / overflow status reporting.

Command $14: Not implemented.

Command $15: Read Quadrature Position. The user may obtain the current angular position with the read quadrature position command. This command is used to obtain position data updates following the issuance of a QPM command ($13) in noncontinuous mode. The current angular position is returned as a 32-bit floating-point value calculated by subtracting the counterclockwise 32-bit count from the clockwise 32-bit count and converting the result to floating-point format using the Quadrature scale factor. This angular position result is stored in the QPM channel control block at offset $14, as shown in Table 3-28 on page 73. The direction flag is set to $FF for clockwise motion, and $00 for counterclockwise motion. For non-error operation, the VMIVME-2540 will respond with a “QPM ready” status to this command, or with a limit / overflow status when those events occur. Minimum response time may be up to 4 ms due to the floating point operations associated with this measurement.

A Read command issued for a channel that has not previously been set up with the “QPM” command ($13) will result in a command status code of *channel allocation error.*

Front Panel Input Signal Connections: The quadrature clockwise signal is connected to the even (lower) channel clock input of the channel pair. The quadrature counterclockwise signal is connected to the odd (upper) channel clock of the channel pair.

Integer Quadrature Position Measurement

Integer quadrature position measurement is very similar to the floating-point QPM, but eliminates the overhead of the software floating-point operations and results in higher data throughput (Table 3-34 below). The two quadrature signals are connected to the clock inputs of two consecutive channels, even and odd. The direction sense of the measurement is determined by the quadrature relationship of the input signals: when the even channel signal input leads the odd channel signal input, the direction is defined to be *clockwise*. The data format for integer QPM is 32-bit signed two's complement with the convention that clockwise rotation is a positive angle and counterclockwise is negative.

Two of the 16-bit counters record the clockwise and counterclockwise pulses generated by the front-end logic. These 16-bit hardware counters are augmented by 16-bit *software counters* using variables in local memory. Absolute position is calculated by subtracting the counterclockwise 32-bit count from the clockwise 32-bit count. In the discrete data mode, the user acquires integer QPM angle data from the counters by sending the Read Integer Quadrature Position command.

Since the quadrature position is not recorded with a hardware *up-down* counter, the possibility exists where jitter at the quadrature encoder shaft causes repetitive direction changes and succeeding trains of clockwise and counterclockwise clock pulses. Each 32-bit count variable would be incremented until overflow occurred. The VMIVME-2540 provides for encoder jitter, normalizing the two 32-bit count variables by subtracting the smaller count from the larger count, and then clearing the smaller count. A further special case occurs when unrestricted rotation (or motion) occurs in a single direction resulting in a 32-bit quadrature overflow. When an overflow occurs, the VMIVME-2540 local CPU writes the corresponding overflow code to CCB offset $01 and queues the VMEbus interrupt for that channel's CCB, if enabled. If the VMEbus interrupt IRQ is not enabled, the *limit alarm* message is posted to the message queue with the corresponding quadrature channel number.

NOTE: Integer QPM current count data is available only via the *Read Integer QPM* command or the continuous measurement mode. The measurement queue and the interrupt events are used only for limit / overflow status reporting.

Table 3-34 Integer QPM Channel Control Block

Offset	Function
$00	Reserved
$01	Channel Status
$02	VMEbus IRQ Level - Overflow interrupt
$03	VMEbus Vector - Overflow Interrupt
$04	Reserved
$08	Reserved
$0C	Reserved
$10	Reserved
$11	Direction Indicator
$12 to $13	Reserved
$14	32-bit Two's Complement QPM Current Count
$18 to $1F	Reserved

Command $16: Integer Quadrature Position Measurement. Integer quadrature position measurement is initiated for a given channel pair by first writing the VMEbus IRQ and vector in the channel control block, if used for overflow / limit status notification. The user next writes the lower (even) channel ID code to the Channel ID variable and issues the QPM command to the VMIVME-2540 local CPU. When the *command acknowledge* command status code is returned, integer quadrature position measurement is active for the indicated channel pair at absolute position zero. Continuous measurement mode is available for this command. In continuous measurement mode, the QPM measurement is acquired by the local CPU and stored in the QPM count location of the CCB (offset location $0014) as rapidly as data acquisition and processing can occur.

Command $17: Read Integer Quadrature Position. This command is used to acquire current integer QPM position data in the discrete mode of operation after the integer quadrature position measurement command has been issued and the command acknowledged. The current angular position is returned as a 32-bit signed two's complement value calculated by subtracting the counterclockwise 32-bit count from the clockwise 32-bit count. This angular position result is stored in the QPM channel control block at offset $14, as shown in Table 3-34 on page 81. The direction flag (offset $11) is set to $FF for clockwise motion, and $00 for counterclockwise motion.

Front Panel Input Signal Connections: The quadrature clockwise signal is connected to the even (lower) channel clock input of the channel pair. The quadrature counterclockwise signal is connected to the odd (upper) channel clock of the channel pair.

16-bit Integer Period Measurement

This measurement mode counts the number of internal clock source periods which occur between consecutive rising (or falling) input signal edges. The input signal is connected to the gate input of the channel. As with other measurement modes, the host may set parameters to select the autoranging mode, or use a specific clock source (200 ns, 2 µs, 20 µs, 200 µs, or 2 ms) for the clock select code. The host program may also configure the channel for continuous measurement mode. The local CPU will acknowledge the command for successful command processing. When a period measurement is subsequently available, the *period measurement ready* code is written to the CCB. If the 16-bit counter overflows, the *scale error* code is written to the CCB. If the channel is not configured for continuous measurement, either the VMEbus IRQ is asserted to the host, or the appropriate status code is posted to the measurement queue and the measurement ready flag $FFFF is set. The channel control block contains a variable (offset $07) which is the clock period code for the clock source used to measure the input signal period in both the autoranging or user select clock modes. This *clock period code* (offset $07) is used by the local CPU to report the clock period used for the measurement in the autoranging mode. The clock period code return matches the codes of Table 3-25 on page 68 which is used by the host to specify autoranging by writing $00 to offset $01. In autoranging mode, a channel status of *under-range* may be returned in the CCB status byte. The 16-bit integer clock will be valid for this status code, and indicates the local CPU is autoranging for a higher resolution for the subsequent clock count measurement. For a rapidly shifting input signal, this status may be followed by an *over-range* status on the subsequent status and measurement update, and indication of an erratic input signal.

The period of the measured data in seconds may be calculated using the clock period code returned in offset $07. The channel control block for Integer Period Measurement contains the parameters shown in Table 3-35 below.

Table 3-35 Integer Period Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Clock Select ($00 through $05)
$02	VMEbus IRQ Level
$03	VMEbus Vector
$04	16-bit Integer Clock Count (No. of Clocks)
$06	Channel Status
$07	Clock Period Code ($01 through $05)
$08 to $0C	Reserved
$0D	Operational Mode Select Flag (page 69)
$0E to $0F	Reserved

Command $20: 16-bit integer period measurement. This measurement is initiated by writing the user-selected parameters for CCB IRQ / Vector (if used), the clock source, selecting the measurement mode (discrete or continuous at $000A) and clearing the channel status. The host program then initiates the standard command send sequence (clear command status, send command) and verifies that the command acknowledged status is received. The measurement is returned to the user at location $0004 as an unsigned integer word followed immediately by the channel status code of *period measurement ready.*

Front Panel Input Signal Connections: The user signal is connected to the gate input of the selected channel.

32-bit Integer Period Measurement

This CCB format is shown in Table 3-36 on page 84, and the measurement data is an 32-bit unsigned integer quantity. This command is generally used to obtain higher resolution measurements on lower frequency input signals, and may also be utilized for very low frequency input signals. This command requires the use of three channels: a single channel to convert the input signal from a single cycle to a pulse width (using frequency divider and selecting two as the divisor). The output of the frequency divider channel is connected to the input of the channel which has been selected by the user for the 32-bit integer period measurement. The channel selected for the measurement must be an even channel, and the even channel and the next higher odd channel are used for the measurement.

The command supports three modes of operation. Report every measurement (mode 0), Average N-measurements (mode 1), and Report each Nth measurement (mode 2). The sample size / discard count is a 16-bit unsigned integer with a range of 0 through 65,535. Byte element five of the CCB specifies the clock period selection to be used for the measurement. The available clock periods (and their selection codes) are listed in Table 3-24 on page 68. No autoranging is available since a minimum time requirement

for the counter terminal count would be approximately 14.3 minutes. If other than the above parameters are used for the mode and clock period selection, a *scale error* code is returned for the command status.

The measurement returns a count of the number of the above selected clocks that occur between rising edges of the gate input signal. The measurement is stored as a long word in the 32-bit integer period measurement beginning with byte 8 of the CCB. The period measurement is an unsigned integer value with a range of 1 through 4,294,967,295.

If the N-sample size average mode is used, the accumulation of measurements is available in the CCB of the next higher odd channel as a 64-bit quantity beginning with byte element 8. Byte element 6 contains the remaining sample count for the average. As with host monitoring of all elements of the VMEbus shared memory, access to these parameters should be paced to avoid causing excessive delay in processing by the VMIVME-2540 local processor.

A *scale error* code will be returned if the cascaded counters reach the maximum count. However, at least one rising edge is required to trigger the counting process, and the scale error will only occur if a rising edge is not followed by a second rising edge within the maximum time period as determined by the clock frequency selected (for the 200 ns clock this is approximately 14.3 minutes).

Table 3-36 32-bit Integer Period Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Reserved
$02	VMEbus IRQ Level
$03	VMEbus Vector
$04	Measurement Mode (0, 1, or 2)
$05	Clock Period Select (1, 2, 3, 4, or 5)
$06	Average Sample Size / Discard Count (N)
$08 to $0B	32-bit Integer Period Measurement
$0C	Channel Status
$0D	Operational Mode Select Flag (page 69)
$0E to $1F	Reserved

Command $27: 32-bit integer period measurement. This measurement is initiated by writing the user-selected parameters for CCB IRQ / Vector (if used), the measurement mode, the clock period select, the sample size (or discard size, if mode 2) to the CCB, followed by writing this command code to the command word at offset location $04. Discrete operation for the acquisition of data requires that the command be sent for each measurement desired. Continuous measurement mode is available for this command and the period measurement data will be updated in the CCB as rapidly as the local processor can acquire the data. A *period ready* code is returned as the channel status as each new measurement is placed in the CCB.

Front Panel Input Signal Connections: The user signal is connected to the clock inputs of any available single channel (odd or even) selected by the user. The outputs of that channel are then connected to the gate input of the even channel of a channel pair selected for the 32-bit integer period measurement. The single channel must be configured by the user for *frequency divider* with a divisor of two. The even channel of the channel pair is configured for 32-bit integer period measurement using the above command code ($27) and the command parameters.

32-bit Integer Pulse-Width Measurement

The elements of the CCB for this mode of operation is shown in Table 3-37 on page 86. This command utilizes two adjacent channels beginning with an even channel number (that is, 0, 2, 4....). The command enables a 32-bit unsigned integer measurement of the user signal placed on the gate input pin of the selected even channel. The user may select a low active level input or a high active level input using byte element 1 of the CCB. A value of 0 selects an active high-level pulse and a value of two selects a active low-level pulse. The selection of the active low-level input requires that the user *daisy chain* his gate input signal to both the even and odd channel for the command to function properly. This command provides the user with the ability to measure the duty cycle of the gate input signal. With the use of four channels, and selecting an active low pulse on two channels and an active high on two channels, a complete set of signal characteristics can be obtained for the input signal, including duty cycle and period attributes, but requires the host software to perform the computations for those parameters.

The measurement returns a count of the number of the above selected clocks between the rising and falling (or falling and rising) edges of the gate input signal. The pulse-width measurement is stored as a 32-bit unsigned integer quantity beginning with offset byte 8 of the CCB with a range of 1 through 4,294,967,295.

The 32-bit integer pulse width command provides the user with three modes of measurement operation. The modes available are as follows: report every measurement (mode 0), average N-measurements (mode 1), and report each Nth measurement (mode 2). The sample size / discard count is a 16-bit unsigned integer with a range of 0 through 65,535. Byte element five of the CCB specifies the clock period to be used for the measurement. The available clock periods (and their selection codes) are the same as those shown in Table 3-42 on page 95. No autoranging is available since a minimum time requirement for the counter terminal count would be approximately 14.3 minutes. If other than the above parameters are selected for the edge control, mode, and clock period, a *scale error* code will be returned to the user in the command status byte.

The 16-bit integer period measurements should be used if the user's application will permit since measurement data throughput will be improved and hardware resource requirements reduced. This command requires the presence of a minimum of three pulses (or periods) to acquire and return a valid measurement.

This command can also return a *scale error* code if an active-going edge is not followed by an inactive-going edge within the time period as determined by the clock period selection.

Table 3-37 32-bit Pulse-Width Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Gate Edge Select Code
$02	VMEbus IRQ Level
$03	VMEbus Vector
$04	Measurement Mode (0, 1, or 2)
$05	Clock Period Select (1, 2, 3, 4, or 5)
$06	Average Sample Size / Discard Count (N)
$08 to $0B	32-bit Integer Pulse-Width Measurement
$0C	Channel Status
$0D	Operational Mode Select Flag (page 69)
$0E to $1F	Reserved

Command $28: 32-bit integer pulse-width measurement. This measurement is initiated by writing the user-selected parameters for IRQ / Vector (if used), the measurement mode, the clock period select, the input edge select, and the sample discard size (if mode 2 is selected) to the CCB, and then writes this command code to the command word at offset location $04. Discrete operation for the acquisition of data requires that the command be sent for each measurement desired. Continuous measurement mode is available for this command, and the pulse-width measurement data will be updated in the CCB as rapidly as the local processor can acquire the data. A *pulse-width ready* code is returned as the channel status as each new measurement is placed in the CCB.

Front Panel Input Signal Connections: The user signal is connected to the gate inputs of the lower (even) channel of the channel pair. If a pulse-width measurement of a low active pulse is required, the user signal must be connected to both the upper and lower (odd) gate inputs of the channel pair.

Group Acquisition Mode (Integer QPM)

Group acquisition mode allows the host to acquire two integer quadrature position measurements simultaneously. The group acquisition command functions the same as two simultaneous *read integer QPM* commands. The host first initializes the two consecutive integer QPM channels. The lower channel of the QPM channel pair must be on a modulo 4-channel boundary (0, 4, 8, etc.). Since the command is used in conjunction with previous configured QPM channels, it does not have an associated CCB.

When the command is issued by the host program, the local CPU will read each QPM channel, place the QPM values in their respective CCBs, and acknowledge the command for a successful operation, or return an error code in the command status

for a command failure. Continuous measurement mode is available for the group acquisition command.

Command $24: Quadrature group acquisition. The host writes the lower QPM channel number to the channel ID word (offset $000A), and then places the command code in the command word (byte) (offset $0004). If no errors occur, the command status will be updated with a *quadrature position measurement ready* ($06) in the command status byte (or an error code, if one is detected).

16-bit Integer Pulse Measurement

This measurement mode counts the number of internal clock source periods which occur between consecutive rising and falling edges of the input signal. The input signal is connected to the gate input of the selected channel. The CCB format is shown in Table 3-38 below. As with other measurement modes, the host may set parameters to select the autoranging mode, or use a specific clock source (200 ns, 2 µs, 20 µs, 200 µs, or 2 ms). The host may also configure the channel for continuous measurement mode. When the measurement is completed successfully (acknowledged by the local CPU), the *pulse-width measurement ready* code is written to the CCB. If the 16-bit counter overflows, the *scale error* code is written to the CCB. If the channel is not configured for continuous measurement, either the VMEbus IRQ is asserted to the host, or the appropriate status code is posted to the measurement queue and the measurement ready flag $FFFF is set. The CCB contains a variable (at offset $07) which is the clock period select code for the clock source used to measure the input signal period in both the autoranging or user-selected clock modes. This *clock period code* is used by the local CPU to report the clock period used for the measurement in the autoranging mode. The clock period code return matches the codes of Table 3-25 on page 68 which are used by the host to specify autoranging by writing $00 at offset $01. The period of the measured data may be calculated by the host program using the clock period code returned in offset $07. The channel control block for Integer Pulse-Width Measurement contains the parameters shown in Table 3-38.

Table 3-38 16-bit Integer Pulse-Width Measurement Channel Control Block

Offset	Function
$00	Reserved
$01	Clock ($00 through $05)
$02	VMEbus IRQ Level
$03	VMEbus Vector
$04	16-bit Integer Clock Source Period Count
$06	Channel Status
$07	Clock Period Code ($01 through $05)
$08 to $0C	Reserved
$0D	Operational Mode Select Flag (page 69)
$0E to $0F	Reserved

Command $26: 16-bit integer pulse-width measurement. This measurement is initiated by writing the user-selected parameters for CCB IRQ / Vector (if used), the

clock source, selecting the measurement mode (discrete or continuous at $000A), clearing the channel status, and writing the channel number to the channel ID. The host program then issues a clear command status, verifies the command status is clear, sends command code $26, and verifies command acknowledgment. The measurement is returned to the user at CCB offset location $0004 as an unsigned integer word when the channel status is updated to *pulse-width measurement ready.*

Front Panel Input Signal Connections: The user signal to be measured is connected to the gate input of the selected channel.

Delayed Event Timer with VMEbus Interrupt

The delayed event timer command allows the host to allocate any of the 24 channels as a 16-bit delay timer triggered by an edge gate signal input. The timer begins the delay at the active edge, and when the timer delay expires the specified IRQ level and IRQ vector are used to send an interrupt to the host. This command can be used as a single event timer or repetitive event using the multiple event flag. If the host sets the multiple event flag, every subsequent active edge on the gate input will restart the delay count. This command also permits the use of the edge to provide only a pulse output (no interrupt) at the end of the delay. This mode is programmed by setting the IRQ level to $FF. The event delay timer operates in a retriggering default mode. This means that if the delay time is greater than the time between successive active edges, the counter will retrigger. This retriggering has the following effect: The first active edge will save the elapsed time in an internal register (not accessible by the user), the second active edge will load the delay counter, and the third active edge will initiate counting. The user will only receive the interrupt on an active edge which is preceded by an edge by the length of time greater than the delay time, and, if the input has caused a retriggering, a total of three edges will have occurred prior to the interrupt delivery (and/or the pulse output). This will have the effect of suppressing an interrupt delivery (and the pulse output) until the edge signal period exceeds the delay time for three successive cycles.

The retriggering feature can be suppressed by using the retrigger suppression flag. Setting this flag to a nonzero value will cause the successive active edges (which are separated by less than the delay time) to be ignored until the delay time has elapsed. After the delay time has elapsed, the succeeding edges will restart the delay count, if the multiple event flag is set. This has the effect of providing the event delay from the first detected active edge after the command is sent from the host.

Note also that due to interrupt overhead and processing, the local CPU can be placed in an infinite loop by nonretriggering input edges which occur at a frequency greater than approximately 12 kHz if the VMEbus IRQ and vector are set for interrupt delivery.

This event delay also provides an update to the channel status byte of the CCB at the end of the delay. For error-free operation, this status will normally be a *timer alarm* status. The timer alarm status will remain in the CCB until acknowledged by the host program (by clearing the status byte). The channel control block for the event counter CCB format is shown in Table 3-39 on page 89.

Table 3-39 Delayed Event Timer CCB Format

Offset	Function
$00	Reserved
$01	Event Clock Edge (zero for falling edge, nonzero for rising)
$02	VMEbus IRQ ($FF for trigger pulse output only)
$03	VMEbus Vector
$04	32-bit IEEE-754 Format Floating-Point Timer Delay (seconds)
$08	Multiple Events Flag (zero for single, nonzero for multiple)
$09	Retrigger Suppression Flag (nonzero to suppress retriggering)
$0A to $0B	Reserved
$0C	Channel Status (*timer alarm* - $08 for normal operation)
$09 to $0B	Reserved

Command $25: The host initiates Delayed Event Timer mode by writing the desired input clock edge, the CCB IRQ and vector (if used), the multiple event flag, the retrigger suppression flag, and the desired delay parameter to the channel control block. After the parameters have been initialized, the host writes the event counter channel ID to the channel variable at offset $000A, and then writes the command value at board offset $0004.

Front Panel Input Signal Connections: The signal which will provide the edge is connected to the gate inputs of the selected channel.

Programming Strategies for Input Operations

The VMIVME-2540 supports several host program strategies for the acquisition of data for the input operations. The strategy used will be dependent on the data acquisition mode selected for channel operation. The data acquisition modes available are either continuous or discrete. The data acquisition mode is selected prior to sending the command (write to location $0004) to the VMIVME-2540. The data acquisition mode is selected by setting ($FF) or clearing ($00) the continuous / discrete flag (location $000B) prior to sending the command.

Continuous Data Acquisition Mode

The continuous data acquisition / measurement mode causes the local CPU to collect measurement data for that channel as rapidly as processing can occur. The update rate for the measurement data in the channel CCB will be related to the number of channels operating in continuous data acquisition mode, and the amount of host accesses to the VMIVME-2540 via the VMEbus.

The following commands are supported under continuous data acquisition mode:

- 16-bit Event Counting with or without level gating
- 32-bit Event Counting with or without level gating
- 16-bit Period / Frequency / Pulse-Width Measurements
- Enhanced 16-bit Period / Frequency / Pulse-Width Measurement
- Quadrature Position Measurement
- Integer Quadrature Position Measurement / Group Acquisition
- 16-bit Integer Period / Pulse-Width Measurements
- 32-bit Integer Period / Pulse-Width Measurements

For the continuous measurement mode, measurement date availability can be detected by the host program by using either of two available methods:

- Polling individual CCB channel status codes
- Polling individual channel data validity flags

The host programs a channel for continuous measurement mode by placing a nonzero value in the continuous / discrete flag, usually $FF, (refer to Table 3-1 on page 51) at byte offset location $000B, selects the channel (by writing the channel number to location $000A), and writes the channel parameters to the channel control block of the selected channel. The *clear command status* command is issued to clear the command status buffer, and the command status is verified as zero by the host program. The command is then issued for the channel measurement operation by writing the measurement command code to the command location at word offset $0004 (byte offset $0005), and the resulting command status should be verified for a command acknowledge code (or the resulting error code should be processed for corrective action). In continuous measurement mode, both the CCB and channel data validity flags are updated by the local CPU when measurement data is written to the CCB. The VMEbus host program may choose to monitor either the CCB channel status code or the data validity flag to detect updates to data measurements in the CCB.

The local CPU clears the measurement data validity flag, updates the CCB measurement data, and sets the data validity flag indicating measurement data availability. The host program may choose to clear the data validity flag after the measurement data is read, which will allow the host to detect the next available updated data measurement in the CCB.

If the CCB channel status byte is used for data acquisition, the host program should clear the channel status prior to writing the command to set up channel operation. The local CPU will update the CCB channel status code any time that an error is detected in channel operation, a limit condition is reached, or a data measurement has been updated. The host program then monitors the CCB status byte to detect a nonzero status condition when it occurs, indicating a change has occurred in channel status. The retrieved code can then be used to detect an error condition, a measurement ready, or a limit condition, each of which may require a separate

processing segment by the host program. The channel status byte should be cleared immediately following the access of a nonzero value in preparation for detecting subsequent channel status changes.

Accesses to either the CCB channel status or the data validity flag require local bus arbitration. The accesses should be paced to avoid adverse impacts to the local CPU processing.

NOTE: Interrupts are not available for continuous measurement mode, and the channel control block IRQ and vector are not used in continuous measurement mode.

Discrete Data Acquisition Mode

Discrete data acquisition mode is used to acquire a single data measurement from a selected channel when a command is issued. The continuous / discrete flag (location $000B) must to set to zero (0) prior to writing the command code to the command register (location $0004; byte $0005). Discrete data acquisition host processing will vary according to the type of input command issued to the VMIVME-2540 as follows:

For event count and quadrature position measurements, channel operation is initialized by sending the command for the event count or quadrature input. The local CPU will configure the counter channel(s) for the commanded operation, initializing counter hardware to perform the input. The discrete measurement data is then acquired for those input modes by sending a *read* command. Sending the *read* commands will result in the local CPU accessing the channel data and placing that data in the CCB (see *Channel Input/Measurement Command Codes* on page 71 for details of these commands). All other input operations (period, frequency, pulse width, etc.) require that the host reissue the command each time a data measurement is needed by the host processor, and that a *disable channel* command be issued after each measurement is acquired (or the *disable channel* command may be issued preceding the measurement command).

The VMIVME-2540 provides three different methods for host notification of input data availability or channel status change. These are as follows:

- Data and channel status change reporting
- The use of interrupts for discrete (noncontinuous) channel measurement, the use of a single-polled status flag to obtain channel status / measurement data availability from all active channels, using the measurement queue contained within the local CPU memory space.
- The CCB channel status for each channel. The channel status is always updated with status information when any channel status change occurs, including availability of measurement data.

The use of interrupts and the use of the measurement queue interface are complementary; that is, if the VMEbus IRQ for the channel is not enabled (or is not used), the corresponding measurement code and channel ID are inserted into the measurement queue. If IRQ level and vector are used (nonzero and enabled) in the channel CCB, then the measurement queue will not be used for measurement reporting to the host.

The interrupt-driven measurement mode uses the VMEbus IRQ and vector stored in the channel control block to notify the user of completion of a measurement operation

and the availability of measurement or status data in the associated channel control block. The user's interrupt service routine should verify that a successful measurement has occurred by examining the channel status byte located in the channel control block. For a successful measurement, the channel status byte will indicate some ...*measurement ready* status; that is, *period measurement ready*, *pulse-width measurement ready*, etc., depending on the command for that channel. This approach will allow the detection of error conditions, which are also reported via the channel measurement interrupt. The interrupts are delivered to the host at a 1 kHz rate, using an internal interrupt queue within the local CPU's memory.

Discrete input operations for event and QPM measurements use the *read...* command. The command status interrupt provides the interrupt interface for these data acquisition. Both the CCB status and the command status will reflect a ...*measurement ready* (or error code) when the operation is completed and the command status interrupt is sent to the host processor.

If the associated IRQ and vector for that channel are zero (or the interrupt is disabled), the local CPU places the channel ID and the channel status in the local memory of the VMIVME-2540 local CPU which is not accessible from the VMEbus. This queue contains the channel status and channel ID information for all input measurements which have been performed by the local CPU and are pending delivery to the host processor. The measurements are placed on the queue sequentially as data is acquired. Hence, the measurement queue may contain multiple entries for the same channel if the host does not service the measurement queue flag at a higher rate than channel measurement data is acquired. This situation can occur if multiple discrete mode commands are sent to the local CPU and the measurement queue flag is not polled and cleared for the measure queue entries. Note also that if the queue is not serviced (via the measurement queue flag) at a rate higher than the data acquisition rate of the local CPU, measure queue overflow can occur with the resulting loss of measurement queue entries. The queue is managed as a circular queue using input and output pointers by the local CPU, and when overflow occurs, existing entries in the queue will be overwritten. No status reporting occurs for a measurement queue overflow, permitting the user to ignore the measurement ready flag (location $000C) if other measurement input strategies (such as monitoring the CCB status byte) are used.

When the local CPU measurement queue is not empty, the VMIVME-2540 local CPU moves the top queue entry, located by the local CPU output pointer, to the measurement channel ID and measurement status (locations $000E and $000F, respectively), and then sets the measurement flag (location $000A) to $FFFF. The measurement queue is serviced at a 1 kHz rate by the local CPU. Single entries are removed from the local processor internal queue and placed in the shared memory area each time the measurement flag is detected as cleared (zeroed by the host processor).

After the host program completes the read of the measurement channel ID and code, the host should clear the measurement flag. Unlike the command status register, host access to the measurement flag is arbitrated by the VMIVME-2540 on a cycle-by-cycle basis. Due to this local bus arbitration, rapid polling of the measurement flag by the VMEbus host may degrade the performance of the VMIVME-2540 local CPU. The VMEbus host should pace access to the measurement flag to avoid adversely affecting the local CPU performance.

Host Processing Sequence: Initialize the channel CCB with the parameters desired for the operation to be performed, set the applicable (either CCB or Command Status) IRQ level and vector (if interrupts are used for measurement ready notification), or

clear the IRQ level and vector location in the channel CCB (if the measurement queue or channel status in the CCB is used for measurement ready notification). Clear the channel status in the CCB. Select the discrete measurement mode by setting the continuous / discrete flag (location $000A) to zero. Send the *clear command status* command, and then verify that the command status code is zero. Write the command code for the desired measurement to location $0004.

If polling is used, wait for a nonzero status code at word location $0006 (byte location $0007), then process the return code (*command acknowledge* - $01 for success or error processing, and notification for other codes (...*measurement ready* will be returned for command status for event and QPM *read* commands). For commands other than *read* operations, read the channel status of the CCB for a nonzero value. When a nonzero is placed in the CCB channel status, the operation has completed and a ... *measurement ready* code is returned, measurement data is available in the CCB (or an error has occurred for the data acquisition and the channel status will identify the type of error).

If interrupts are used for command servicing, an interrupt service routine should process the command status return code for both successful (*command acknowledge*) and command error status codes.

For interrupt notification using the *read*... commands, the data will be available (or error status information) with the command status interrupt delivery. For all other input operations, the data will be available after the local CPU acquires the data and places it in the CCB measurement location. The CCB channel status is updated when the measurement data is placed in the CCB. The CCB IRQ level and vector will be used to interrupt the host. All interrupt service routines for data acquisition should clear the channel status in the CCB in preparation for subsequent data delivery via the interrupt interface.

If the measurement queue is used, read the measurement queue flag and test for $FFFF. If the measurement queue flag is $FFFF, read the measurement channel and status bytes, moving them to host CPU memory space for processing and then clear the measurement queue flag in preparation for further measurement queue data updates from the local CPU.

Channel Output/Waveform Generation Command Codes

16-bit Frequency Divider

The parameter list for the 16-bit frequency divider CCB contains a 16-bit binary integer divisor supplied by the user prior to the command interrupt. This CCB format is shown in Table 3-45 on page 97. Frequency division is accomplished by connecting the frequency source to the clock input of the indicated channel and extracting the divided signal from the output signal of the same channel. The output signal generated by the frequency will be a square wave with a 50 percent duty cycle and independent of the clock input duty cycle. This command will generate a continuous output signal until a *disable channel* command is received by the local CPU.

Table 3-40 16-bit Frequency Divider Channel Control Block

Offset	Function
$00	Reserved
$04	16-bit Divisor (Binary Integer)
$06 to $0F	Reserved

Command $07: 16-bit Frequency Divider. Range of divider is 2 through 65,535. A *bounds error* channel status will be returned for divider values less than 2.

Front Panel Input Signal Connections: The signal to be divided is connected to the clock input of the selected channel. The divided signal is available at the output pin of the selected channel.

32-bit Frequency Divider

A 32-bit frequency division is accomplished by connecting the frequency source to the clock input of the lower (even) channel and extracting the divided signal from the output signal of the next higher (odd) channel. For successful operation, the output signal of the higher channel must be connected externally to the gate input of the lower channel. The parameter list for the 32-bit frequency divider CCB contains a 32-bit binary integer divisor supplied by the user prior to the command interrupt. This CCB format is shown in Table 3-41 below. The 32-bit frequency division may be commanded for even channels only: channels 0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, and 22. If the user commands a 32-bit frequency division for an odd channel number, the VMIVME-2540 local CPU gives the *channel allocation error* command status code. For a channel pair to be successfully set up for 32-bit frequency division, the next higher channel must first be disabled. If the user commands 32-bit frequency division for a channel pair which has not been previously disabled, the local CPU gives the *active channel error* command status code. The 32-bit divisor must be a quantity greater than 65,536. If the 32-bit divisor is 65,536 or less, the local CPU gives the *bounds error* command status code. The frequency divider channel will generate a continuous output signal, based on the input signal, until a *disable channel* command is received by the local CPU.

Table 3-41 32-bit Frequency Divider Channel Control Block

Offset	Function
$00	Reserved
$04	32-bit Divisor (Binary Integer)
$08 to $0F	Reserved

Command $08: 32-bit Frequency Divider. This enhanced 32-bit mode of operation **requires that the user externally connect the output signal from the next higher channel N+1 to the gate input of channel N**. Once the counters have been initialized, the command status buffer is loaded with the *divider-ready* message, and the VMEbus command status IRQ and vector are placed in the VMEbus interrupt queue, if the interrupt is enabled and the IRQ level is nonzero.

Front Panel Input Signal Connections: The signal to be divided is connected to the clock input of the lower (even) channel of the channel pair. The divided signal is available on the upper (odd) channel of the channel pair. The outputs (high and low) of the upper channel must also be connected to the gate inputs (high and low) of the lower (odd channel).

Period/Pulse-Width Generation

The parameter list for the period / pulse-width generation mode contains a 32-bit IEEE-754 floating-point period value, and a 32-bit IEEE-754 floating-point pulse width value. This CCB format is shown in Table 3-42 below. The commanded period must fall within the range of 400 ns to 262.14 s for both square-wave generation and pulse train generation. For commanded periods outside these ranges, the local CPU returns the *bounds error* command status code. The commanded pulse width must fall within the range of 200 ns to 131.07 s, or the local CPU will return the *bounds error* command status code.

A *disable channel* command between subsequent writes of the period / pulse-width generation command is not required. If a channel has been previously set up for generation, the host program may command new (different) period values as required. A channel operating in this mode will generate a continuous signal until a *disable channel* command is received by the local CPU.

NOTE: The maximum pulse width high or low is 131.07 seconds, limiting the duty cycle for periods between 131.07 to 262.14 seconds.

Table 3-42 Period/Pulse-Width Generation Channel Control Block

Offset	Function
$00 to $03	Reserved
$04	32-bit IEEE-754 Floating-Point Period Value (seconds)
$08	32-bit IEEE-754 Floating-Point Pulse-Width Value (seconds)
$0C to $0F	Reserved

Command $09: Square-Wave Generation, 16-bit counter with internal time base and a 50 percent duty cycle. The pulse width value is not used.

Command $0A: Pulse Train Generation, 16-bit counter with internal time base. The period value is scaled to choose an internal time base and total counter period. The pulse width value will determine the output high duration and the difference between the period value, and the pulse width value will determine the output low duration. The use of a pulse width parameter which is greater than the period parameter will result in an error return of *period error* and an output signal which has the specified pulse width period of 13.2 milliseconds.

Front Panel Input Signal Connections: None.

Frequency/Duty Cycle Generation

The parameter list for the frequency / duty cycle generation mode contains a 32-bit IEEE-754 floating-point frequency value, and a 32-bit IEEE-754 floating-point duty cycle value. This CCB format is shown in Table 3-43 below. The commanded frequency value must fall between the values of 2.5 MHz and .007629 Hz for square-wave generation, 2.5 MHz to 0.003815 Hz for pulse train generation. For commanded frequencies outside these ranges, the VMIVME-2540 local CPU returns the *bounds error* command status code. Duty cycle values are expressed as percent, 0 to 100 percent. The smallest duty cycle is determined by the ratio of 200 ns to the reciprocal of the commanded frequency since the smallest pulse width must be one 5 MHz clock cycle. Similarly, the largest duty cycle is determined by the ratio of the period of the commanded waveform less 200 ns to the commanded period.

A *disable channel* command is not required between subsequent writes of the frequency / duty cycle generation command. If a channel has been previously set up for frequency / duty cycle generation, the host may command new (different) frequency values as required by the host program application. A channel operating in this mode will generate the signal commanded until a *disable channel* command is received by the local CPU.

Table 3-43 Frequency/Duty Cycle Generation Channel Control Block

Offset	Function
$00 to $03	Reserved
$04	32-bit IEEE-754 Floating-Point Frequency Value (Hz)
$08	32-bit IEEE-754 Floating-Point Duty Cycle Value (Percent)
$0C to $0F	Reserved

Command $0B: Frequency Generation. 16-bit counter with internal time base. The frequency value is scaled to choose an internal time base and total counter period. The output signal is a 50 percent duty cycle. The duty cycle parameter is not used.

Command $0C: Duty Cycle Generation. 16-bit counter with internal time base. The frequency value is scaled to choose an internal time base and total counter period. The duty cycle value is then used to obtain the output high pulse width from the total counter period. The duty cycle is entered as a quantity between 0.00 and 100.00.

Front Panel Input Signal Connections: None.

Pulse Sequence Generation

This output mode allows the VMEbus host to command the VMIVME-2540 to generate a finite sequence of pulses with 0 to 100 percent duty cycle. Pulse sequence mode requires two adjacent channels: an even channel N which generates the pulses and an odd channel N +1 which counts the pulses. External wires are required on the P3 / P4 front panel connect from the output of channel N+1 back to the gate input of channel N. Pulse sequence mode may only be commanded for even-numbered channels. The two channels must be disabled before the first pulse sequence command is issued. Consecutive pulse sequence commands may be sent for a selected channel without sending a *disable channel* command between them. See Table 3-44 on page 97.

Table 3-44 Pulse Sequence CCB Parameters

Offset	Function
$00	Reserved
$01	Clock Period Select ($01 through $05)
$02	VMEbus IRQ
$03	VMEbus Vector
$04	16-bit Integer Number of Clock Cycles Output Low
$06	16-bit Integer Number of Clock Cycles Output High
$08	16-bit Integer Number of Pulse
$0A to $0E	Reserved
$0F	Completion Flag

Command $21: The host initializes the clock select (codes 1 through 5, Table 3-24 on page 68), number of clocks high (high-level duration), number of clocks low (low-level duration), and the number of pulses (or cycles) (1<N<32767) to be output. For interrupt-notification of the operation completion, the host initializes the VMEbus IRQ and vector locations in the CCB. The host selects the channel by writing the channel number to the channel ID location, then writes $21 to the command location at offset $0004. After pulse train completion, the VMIVME-2540 local CPU writes $FF to offset $0F in channel status of the CCB. If the VMEbus IRQ is not enabled for the channel, the VMEbus *limit alarm* is placed in the message queue after completion of the pulse train. The channel control block for pulse sequence mode contains the parameters shown in Table 3-44 above.

Front Panel Input Signal Connections: None.

Programmed Output Mode

This output mode allows each channel output of the VMIVME-2540 to operate as a discrete digital output with the host program controlling the output state. The host program sets the digital output state by setting the state parameter ($00 for a zero volt output signal or $01 for a 5 volt output level) in the CCB and sending the programmed output command. Transport lag for local CPU processing before a changed output state occurs is approximately 60 µs.

Table 3-45 Programmed Output CCB Parameters

Offset	Function
$00	Reserved
$01	Output State - 0 for Logic Low, 1 for Logic High
$02 to $0F	Reserved

Command $22: This command sets the channel output to the selected state. The host writes the desired logic state (1 or 0) to offset $01 in the channel control block, initializes the channel ID location, and then writes $22 to the command location. When the VMIVME-2540 local CPU has set up the output, the command acknowledge status code is returned. See the channel control block, Table 3-45 above, for programmed output mode contains the desired output state. The channel output state

is set to the selected level by the local CPU immediately prior to the *command acknowledge* status return.

Front Panel Input Signal Connections: The programmed output is available on the output pins of the selected channel.

Quadrature Position Control

This output mode allows the VMEbus host to command the VMIVME-2540 to generate sequence of pulses on two channels in quadrature mode. The host writes the desired signed quadrature position change (delta) in the channel control block (CCB), while the VMIVME-2540 local CPU maintains the current absolute quadrature position in the CCB. The VMIVME-2540 CPU configures the phase of the two quadrature signals using the sign of the 16-bit two's complement change variable. When the new position is attained, either the CCB IRQ is asserted to the host if enabled, or the *quadrature position reached* status code is posted to the measurement queue for polling mode.

Quadrature position control requires a total of four adjacent channels: two adjacent channels for each quadrature signal to be generated. In addition, each channel pair requires external wires routed from the output of channel N+1 back to the gate input of channel N, and the output of channel N+3 back to the gate input of channel N+2. The quadrature output signals are generated by the outputs of channels N and N+2. The quadrature output signals have a 50 percent duty cycle with ±1/4 wave phase mode.

When the quadrature pulse train is completed, the VMIVME-2540 local CPU writes the completion flag $FF to offset $0F in the CCB channel status. If the VMEbus CCB IRQ is enabled for the channel, the VMEbus IRQ is asserted. Note that the command status is used to deliver the *command acknowledge* (or command error) status, and that the CCB IRQ is used to deliver the completion of the operation. When the VMEbus IRQ is not enabled, the local CPU posts the *limit alarm* to the message queue after completion of the quadrature pulse train. The quadrature current position is maintained in the channel CCB. The channel control block for the Quadrature Output Mode contains the parameters shown in Table 3-46 below.

Table 3-46 Quadrature Control Output Mode

Offset	Function
$00	Reserved
$01	Clock Source ($01 through $05)
$02	VMEbus IRQ
$03	VMEbus Vector
$04	16-bit Integer Number of Clock Cycles High (Pulse-Width)
$06	16-bit Integer Two's Complement Quadrature Position Delta
$08	32-bit Integer Two's Complement Current Absolute Position (Number of Clock Cycles)
$0C to $0E	Reserved
$0F	Channel Status

Command $23: The host initializes the clock period select, number of clocks high, number of quadrature pulses to be output (delta), the current absolute position, clears the channel status, writes the VMEbus IRQ, and vector (nonzero for interrupt handling, zero for polling) in the channel CCB. The host then writes the channel

number to the channel ID location, then writes command $23 to offset $0004. It is not necessary to issue the *disable channel* command between subsequent writes of the quadrature position control command when position changes are updated by sending the command with a delta change.

Front Panel Input Signal Connections: For each channel pair, connect external wires from the output of channel N+1 to the gate input of channel N, and the output of channel N+3 to the gate input of channel N+2. The quadrature output signals are available on the outputs of channels N and N+2.

Programming Strategies for Output Operations

The user may select interrupt operation or polling for the output commands of the VMIVME-2540. All digital signal output operations operate in a continuous mode after the commands have been sent and acknowledged by the local CPU with the exception of the pulse sequence and quadrature position control which generate a single sequence of output signals. For host program polling, the host program should initialize the CCB parameters for the selected output operation to include clearing of the CCB channel status, if used. The channel ID should be set for the desired channel. The command should then be written to location $0004. The command status should then be monitored for a *command acknowledge* command status code. When the *command acknowledge* status is received, the output signal is active. Where applicable, the host program should then poll the CCB status for the completion of those operations which operate in a single sequence mode (quadrature position and pulse sequence) if this information is required for host processing.

For interrupt processing for output operations, the command status interrupt handler may be used for the handling of command status after commands are sent using the command status IRQ and vectors. In addition, the interrupt handler may be used for processing the completion of some of the output operations (pulse sequence and quadrature position control) if the host program requires notification of those events. The processing sequence for setting up the output signals would parallel the processing for using the polling methodology, except the polling steps for command status and CCB channel status for operation completion would be replaced with interrupt handling routines.

Timer Operation Command Codes

The VMIVME-2540 provides timer functions based either on internal clock frequencies using timer channel control blocks (TCCBs), or timing delays for a front panel input signal using the standard channel control block (CCB). The following sections describe the timing operations supported by the board. The timer operation's programming strategies are consistent with the channel output programming discussed in the *Programming Strategies for Output Operations* on page 99.

Timer/Periodic Interrupt

The parameter list for the timer / periodic interrupt mode contains the VMEbus IRQ level and interrupt vector to be asserted after the indicated time interval, and a 32-bit IEEE-754 floating-point time interval of seconds. This command uses only the timer channel control block for operation. This TCCB format is shown in Table 3-47 on page 100. The timer / periodic interrupt command may be issued only for timer channels available according to the board channel count; if the user issues commands for timer

channels exceeding the timer channel count of the board, the VMIVME-2540 local CPU will return the *channel allocation error* command status code. A single timer channel is available for each group of four channels, and will vary according to the board option, from one for a 4-channel board to five for a 24-channel board.

NOTE: Timer Channel 0 is used internally to schedule the delivery of interrupts and measurement queue data to the host program. The timer interrupt is available to the user for this channel; however, the interrupt interval is restricted to 1 ms.

Table 3-47 Timer/Periodic Interrupt Channel Control Block

Offset	Function
$00	Reserved
$02	VMEbus IRQ
$03	VMEbus Vector
$04	32-bit: IEEE-754 Floating-Point Time Interval (seconds)
$08 to $0F	Reserved

Command $18: 16-bit Counter, single cycle. The VMIVME-2540 local CPU configures the selected channel with the 200 ns time base for interrupt intervals from 200 ns to 13.1 ms, the 2 µs time base for periods from 13.1 ms to 131 ms, 20 µs time base for periods from 131 ms to 1.31 s, 200 µs time base for periods from 1.31 to 13.1 s, and 2 ms for periods from 13.1 s to 131 s. The VMEbus interrupt is asserted only once for this command.

Command $19: 16-bit Counter, automatic restart. This command contains the same parameter interrupt setup as command $18, but the timer will automatically restart after each VMEbus IRQ has been asserted.

NOTE: The use of a restart timer interrupt with interval of less than approximately 50 µs (The processor data transport time for interrupt delivery) will result in the processor being in a compute-bound condition, and preclude further measurement reporting and command responses from the board.

Command $1A: Disable Timer. This command instructs the VMIVME-2540 local CPU to disarm the counter associated with the indicated timer and halt the VMEbus interrupts associated with that timer.

Front Panel Input Signal Connections: None (no front panel inputs / outputs are available for the timers).

Auxiliary Commands

The VMIVME-2540 has two commands which are not related to channel input, output, or timer functions and have no corresponding channel control blocks in shared memory.

Block Move: Command $1D - The Block Move command instructs the VMIVME-2540 local CPU to move a block of memory, 16 bits at a time, located at 32-bit <start_address> with 16-bit words <qty_words> to a 32-bit <dest_address>. These user parameters are first stored in the diagnostic buffer according to Table 3-48 below. Note that both <start_address> and <dest_address> may point to the VMIVME-2540 local CPU memory: EPROM, I/O addresses, and static RAM. Refer to *Local Address Decode* on page 26 for a description of the VMIVME-2540 local CPU address space.

Table 3-48 Block Move Diagnostic Buffer Entries

Variable Name	Offset
Start_Address	$0400
Dest_Address	$0404
Qty_Words	$0408

Execute: Command $1E - The Execute command instructs the VMIVME-2540 local CPU to simply jump to <dest_address> and resume execution. Note that <dest_address> is a 32-bit variable. The transfer to the <dest_address> is treated as a jump subroutine operation, and a return to normal operation can be accomplished with a M68000 *RTS* (return from subroutine) instruction. Use by the customer is not recommended. The diagnostic buffer entry is shown in Table 3-49 below.

Table 3-49 Diagnostic Buffer Entry for Execute Command

Variable Name	Offset
Dest_Address	$0400

Echo PC: Command $1F - The Echo PC command is a local CPU diagnostic technique and causes the local CPU to return its program counter value at the time the command interrupt occurred. The only situation in which the VMIVME-2540 local CPU will not respond to this command is when a fatal exception has occurred, the interrupts have been disabled, the front panel LED is on, and the 68HC000 CPU has just executed the STOP instruction. Use by the customer is not recommended.

Getting Started

The VMIVME-2540 Intelligent Counter / Controller board provides multiple input and output functionality, which leads to some complexity in its application. This section discusses the host program issues in the use of the VMIVME-2540 for waveform measurement, quadrature position control, and quadrature position measurement with only connections on the front panel. A digital storage oscilloscope is suggested for viewing waveforms during the demonstration. The most inexpensive way to make the interconnections on the VMIVME-2540 front panel is to use a female DIN 96-pin connector with wirewrap connections. A more appropriate method for integrating the VMIVME-2540 into a system is to use the VMIACC-BT04 Dual 96-pin Transition Panel, a 19-inch rack-mountable panel which breaks all 192 signals out from two DIN 96-pin connectors to standard terminal blocks. Connection between the VMIACC-BT04 and the VMIVME-2540 front panel is made by 96-conductor ribbon cables, three foot length recommended. See Section 5.4 for more application data.

The example code will set up the VMIVME-2540 for the configuration shown in Table 3-50 below for example set 1 and the configuration shown in Table 3-51 below for example set 2. Note that example set 2 expects an optional external input for the QPM measurement, but the other features used in that example are provided by the code.

Table 3-50 VMIVME-2540 Configuration Using Example Set 1

Mode	Channels
Quadrature Position Control	0, 1, 2, 3
Quadrature Position Measurement	4, 5
Integer Period Measurement	6
Integer Period Measurement	7

Table 3-51 VMIVME-2540 Configuration Using Example Set 2

Mode	Channels
Pulse Width Output	1
Frequency Divider	0
Event Counter	3
Floating Point Period Measurement	2
Quadrature Position Measurement	4, 5
Quadrature Position Control**	0, 1, 2, 3
NOTE: ** Optional selection with Channels 0 through 3 for this operation.	

For example set 1, the channel 6 integer period measurement will use the same clock frequency (which sets the time base) as the QPC measurement channel. The channel 7 integer period measurement will use the next higher clock frequency. For example set 2, channel zero output is connected to the channel one gate input and channel one output is connected to the channel two gate input. Table 3-52 on page 103 presents the connections for example set 1 and Table 3-53 on page 103 presents the connections for example set 2.

Note that in example set 1 configuration, the quadrature position control output signals (channels 0 and 2) are looped back to the quadrature position measurement inputs (channels 4 and 5), and that one of the quadrature output signals is also routed to period measurement inputs on channels 6 and 7. Since an RS-422 output is being routed to multiple RS-422 line receivers, termination resistor jumpers E36 and E39 must be removed for proper operation. Jumper shunt E27 is left plugged in to terminate the line.

If the above RS-422 termination precautions are observed, the same connector used for example set 1 can be used for example set 2, and adding four additional connections to satisfy connections for both examples with parallel connections for example set 2. If external quadrature signals are used, then a separate connector is required for example set 2.

Table 3-52 Wire Connections - Example Set 1

From Signal Name	From Pin Number	To Pin Number	To Signal Name
Out 1+	P3 - C4	P3 - B1	Gate 0+
Out 1-	P3 - C5	P3 - B2	Gate 0-
Out 3+	P3 - C9	P3 - B7	Gate 2+
Out 3-	P3 - C10	P3 - B8	Gate 2-
Out 0+	P3 - C1	P3 - A12	CLK 4+
Out 0-	P3 - C2	P3 - A13	CLK 4-
Out 2+	P3 - C7	P3 - A15	CLK 5+
Out 2-	P3 - C8	P3 - A16	CLK 5-
CLK 4+	P3 - A12	P3 - B17	Gate 6+
CLK 4-	P3 - A13	P3 - B18	Gate 6-
Gate 6+	P3 - B17	P3 - B20	Gate 7+
Gate 6-	P3 - B18	P3 - B21	Gate 7-

Table 3-53 Wire Connections - Example Set 2

From Signal Name	From Pin Number	To Pin Number	To Signal Name
Out 1+	P3 - C4	P3 - A1	CLK 0+
Out 1-	P3 - C5	P3 - A2	CLK 0-
Out 0+	P3 - C1	P3 - A9	CLK 3+
Out 0-	P3 - C2	P3 - A10	CLK 3-
Out 3+	P3 - C9	P3 - B7	Gate 2+
Out 3-	P3 - C10	P3 - B8	Gate 2-
Ext. Quadrature	CW-Out+	P3 - A12	CLK 4+
Ext. Quadrature	CW-Out-	P3 - A13	CLK 4-
Ext. Quadrature	CCW-Out+	P3 - A15	CLK 5+
Ext. Quadrature	CCW-Out-	P3 - A16	CLK 5-

The header file and the host program, including the main routines and supporting subroutines are shown below.

```
/*************************************************************************/
/*t2540.h   VMIVME-2540 Data Structures            */
/*************************************************************************/
#define     USER_VECTOR_0     100   /* VMEbus vector - 1st user vector */
#define     USER_VECTOR_1     101   /* VMEbus vector - 2nd user vector */
#define     USER_VECTOR_2     102   /* VMEbus vector - 3rd user vector */
#define     USER_VECTOR_3     103   /* VMEbus vector - 4th user vector */
#define     USER_VECTOR_4     104   /* VMEbus vector - 5th user vector */
#define     INTR_ENABLED      8    /* VMIVME-2540 interrupt enabled bit */

/* Interrupt priority levels/VME bus */

#define     LEVEL_1           1
#define     LEVEL_2           2
#define     LEVEL_3           3
#define     LEVEL_4           4
#define     LEVEL_5           5
#define     LEVEL_6           6
#define     LEVEL_7           7

/* Define the available command code set - rev 1.23 and later */

#define     disable_ch        0x00
#define     event_16          0x01
#define     event_lvl_16      0x02
#define     event_32          0x03
#define     event_lvl_32      0x04
#define     event_lim_32      0x05
#define     read_count        0x06
#define     divider_16        0x07
#define     divider_32        0x08
#define     square_wave       0x09
#define     pulse_train       0x0A
#define     freq_gen          0x0B
#define     duty_cycle        0x0C
#define     peri_meas_16      0x0D
#define     peri_meas_32      0x0E
#define     freq_meas_16      0x0F
#define     freq_meas_32      0x10
#define     puls_meas_16      0x11
#define     puls_meas_32      0x12
#define     qpm               0x13
#define     qpm_index         0x14
#define     read_qpm          0x15
#define     qpm_int           0x16
#define     read_qpm_int      0x17
#define     timer_16          0x18
#define     timer_16r         0x19
#define     disable_tmr       0x1A
#define     initialize        0x1B
```

```
#define     reset_resp        0x1C
#define     block_move        0x1D
#define     jump_to           0x1E
#define     echo_pc           0x1F
#define     int_peri_meas     0x20
#define     puls_seq          0x21
#define     output            0x22
#define     qpc               0x23
#define     group_acquire     0x24
#define     event_timer       0x25
#define     puls_meas_int16   0x26
#define     peri_meas_int32   0x27
#define     puls_meas_int32   0x28

/* Define the command status codes and CCB status code */
#define     null              0x00
#define     command_ack       0x01
#define     count_ready       0x02
#define     period_ready      0x03
#define     freq_ready        0x04
#define     width_ready       0x05
#define     qpm_ready         0x06
#define     limit_alarm       0x07
#define     timer_alarm       0x08
#define     ch_alloc_err      0x09
#define     bounds_err        0x0A
#define     period_err        0x0B
#define     pulse_err         0x0C
#define     freq_err          0x0D
#define     scale_err         0x0E
#define     qpm_err           0x0F
#define     gate_err          0x10
#define     limit_err         0x11
#define     ch_active_err     0x12
#define     req_denied        0x13

#define     TRUE              0xFFFFFFFF
#define     FALSE             0
struct vmivme_2540_regs {
    unsigned short bid; unsigned short rev; unsigned short cmd;
    unsigned short response; unsigned char resp_irq; unsigned char resp_vec;
    unsigned char channel; unsigned char contdisc; unsigned short mflag;
    unsigned char mchan; unsigned char mcode;
struct CH_CB {
    unsigned char command; unsigned char gate_edge; unsigned char vme_irq;
    unsigned char vme_vec;
    union {
```

```
    unsigned char cp_b[12]; unsigned short cp_w[6]; unsigned int cp_l[3];
    float cp_f[3];
    } cp;
    } ch_ccb[24];
struct TMR_CB {
    unsigned char tcmd; unsigned char tgate; unsigned char vme_irq;
    unsigned char vme_vec;
    union  {
    unsigned char tp_b[12]; unsigned short tp_w[6]; unsigned int tp_l[3];
    float tp_f[3];
    } tp;
    } tmr_ccb[6];
    unsigned char cd[24]; unsigned char flg[24];
};
typedef struct vmivme_2540_regs ICC;
/* Change the following vector to satify user configuration requirements */

/*******************************END OF t2540.h*************************/


/*************************************************************************
*       File:      gs.c     9/10/01          VMIC                        *
*                                                                        *
*     100 Hz Periodic Interrupt using Timer1                             *
*   Example Set 1                                                        *
*     Quadrature Position Control on channels (0,1,2,3)                  *
*     Integer Quadrature Position Measurement channels (4,5)             *
*     Integer Period Measurement w/200 ns clock on channel (6) measuring *
*     QPC pulse width.                                                   *
*     Integer Period Measurement w/20 us clock on channel (7) measuring  *
*   Example Set 2                                                        *
*     Pulse train output on channel 1                                    *
*     16-bit frequency divider on channel 0                              *
*     16-bit event count on channel 3                                    *
*     Floating point period measurement on channel 2                     *
*     Integer Quadrature Position Measurement on channels (4,5)          *
 *    with optional Quadrature Position Control (internal) on channels   *
*     0,1,2,3(previous channels reconfigured).                           *
*************************************************************************/

#include <stdio.h>
#include <string.h>
#include "t2540.h"
/* The following routine allows the user to input a floating point number into a float variable
gfp. */
#define   getlong(a)   scanf      ("%8x", a)
#define   getbyte(a)   scanf      ("%2d", &int_in); a = (unsigned char )int_in
#define   getword(a)   scanf      ("%6d", &int_in); a = (short )int_in
#define   getfp(a)     scanf      ("%12f", &a)
#define   prompt(a)    printf     ("%s", a); scanf("%c", &byte_in);
ICC * brd = ((ICC *)(0xFB000000));   /* make VMEbus pointer */
char anykey[] = "\r\nPress any key to continue (x to terminate sequence)...";
```

```
unsigned short req_err        /* Interrupt detected command status error */
      resp_flg,               /* command status code */
      word_in;                /* 16-bit variable for user input */
unsigned char ccb_status,
      byte_in;                /* byte variable for user input */
float float_in;/* float variable for user input */
unsigned int int_in;          /* scanf input long int */
unsigned int time,            /* real-time clock ephemeris */
          timeout,            /* timeout after five seconds */
          ch0_cnt,            /* channel 0 ISR count semaphore */
          ch0_old,            /* channel 0 previous count */
          ch2_cnt,            /* channel 2 ISR count semaphore */
          ch2_old,            /* channel 2 previous count */
          ch4_cnt,            /* channel 4 ISR count semaphore */
          ch4_old;            /* channel 4 previous count */
void ex_set_1();
void ex_set_2();
void qpc_setup();
void com2540();
int init2540();

short qpc_delta;
unsigned short qpc_clks;
char qpc_flag;                    /* qpm using internal(=1) or external (=0) */
unsigned char scale;              /* qpc/qpm clock period select code storage */

void
main (void)
{
      unsigned char set_select;
      if (init2540()) {
           printf ("...Exiting.");
           return;
      }
ex_sel:
qpc_clks = 0;                 /* initialize qpc clk count */
      qpc_delta = 0;          /* initialize qpc delta clk count */
      scale = 0;              /* initialize qpc clock select */
      printf ("\r\nSelect example set to demonstrate (1 or 2): ");
      getbyte (set_select);
      if (set_select == 1) ex_set_1();
      else if (set_select == 2) ex_set_2();
      else {
       printf ("\r\nUnknown example set...");
      }
      prompt("\r\nEnter x to exit...any other key to re-select :");
      if (byte_in != 'x') goto ex_sel;
      brd->resp_vec = 0;      /* clear vector */
```

```
brd->resp_irq = 0;      /* clear level */
com2540 (initialize,0);
printf ("\r\nProgram terminating\r\n");
}

void
ex_set_1 (void)
{
    unsigned int q5, told;
    unsigned short p6, p7;
    unsigned char s6, s7, pscale, pclock;
brd->contdisc = 0x0FF;  /* CONT. MEAS. MODE */
com2540 (disable_ch, 4); /* issue a disable command */
com2540 (qpm_int, 4);   /* Integer QPM on channel 4,5 */
pscale = 0;             /* Set previous clock select */
while (TRUE) {
    printf ("\r\nEnter clock select code (1-5): ");
    getbyte (scale);
    brd->ch_ccb[0].gate_edge = scale;
    if (scale != pscale) {
    pclock = scale;
       if (pclock == 1) pclock = 2;       /* limit to lowest clock select */
       com2540 (disable_ch, 6);           /* disable channel 6 */
       brd->ch_ccb[6].gate_edge = pclock-1;   /* use lower clock/ch6*/
       com2540 (int_peri_meas, 6);        /* Integer period meas on ch6 */
       com2540 (disable_ch, 7);           /* disable channel 7 */
       brd->ch_ccb[7].gate_edge = scale;  /* use same clock as QPC/ch7*/
       com2540 (int_peri_meas, 7);        /* Integer period meas on ch7 */
       pscale = scale;                    /* save this select */
    }
qpc_setup();
told = time;
printf ("\r\n\n <4,5>  <6p> <6s> <7p> <7s>\n");
while (!kbhit()) {                         /* terminate when keyboard hit */
      if (time > told ) {                  /* if timer ticked, */
      told = time;                         /* read the clock  */
      q5 = brd->ch_ccb[5].cp.cp_l[0];      /* read all measurements */
      p6 = brd->ch_ccb[6].cp.cp_w[0];
      s6 = brd->ch_ccb[6].cp.cp_b[2];
      p7 = brd->ch_ccb[7].cp.cp_w[0];
      s7 = brd->ch_ccb[7].cp.cp_b[2];
      printf ("\r%.8X %.4X  %.2X  %.4X %.2X", q5,p6,s6,p7,s7);
      }
   }
prompt(anykey);
if (byte_in == 'x') {
      com2540 (disable_ch, 0);
      com2540 (disable_ch, 4);
      break;
```

```
      }
    qpc_clks = 0;    /* initialize qpc clk count */
    qpc_delta = 0;    /* initialize qpc delta clk count */
    }
    return;
}
char qpm_option[] = "\r\nSelect QPM signal source "
             "internal (0) or external (1): ";
void
ex_set_2( void )
{
    unsigned char i, per_clk;
    unsigned short per_siz;

/* Example of Pulse Train Generation:
   the user is prompted for period and pulse width, then
   channel 1 is setup to generate the waveform.
   The signals generated are OUT1+ at P3-C1 and OUT1- at P3-C2.

*/
   com2540 (disable_ch, 1); /* disable channel before next command */
   printf ("\r\nDemonstrating Pulse Train Generation on Channel 1 ...");
   printf ("\r\nEnter Float Period value: ");
   getfp (float_in);
   brd->ch_ccb[1].cp.cp_f[0] = float_in;
   printf ("\r\nEnter Float Pulse Width: ");
   getfp (float_in);
   brd->ch_ccb[1].cp.cp_f[1] = float_in;
   com2540 (pulse_train, 1);
   printf("\r\nPulse train active on Ch. 1.\r\n");
   prompt(anykey);
   if (byte_in == 'x')return;

/* Example of Frequency Division:
   the user is prompted for a 16-bit hex divisor, then channel 0 is
   configured for frequency division. Since channel 1 is already
   generating a pulse train, it may be convenient for the user to
   connect the output of channel 1 to the clock input of channel 0, then
   observe the resultant waveform at OUT0+ at P3-C1.
*/
   com2540 (disable_ch, 0); /* always disable channel before next command */
   printf ("\r\nDemonstrating Frequency Division on Channel 0 ...");
   printf ("\r\nEnter 16-bit divider: ");
   getword (word_in);
   brd->ch_ccb[0].cp.cp_w[0] = word_in;
   com2540 (divider_16, 0);
   prompt (anykey);
   if (byte_in == 'x')return;

/* Example of 16-bit Event Counter:
   count events up to a user-defined limit, then display message.
   Current count displayed whenever a key is pressed. Event signal
   inputs are CLK3+ at P3-A9 and CLK3- at P3-A10.
*/
   com2540 (disable_ch, 3); /* disable channel before next command */

/*Note: Initialize all CCB parameters after each disable command. The
   CCB is cleared with a disable command */
   brd->ch_ccb[3].vme_vec = USER_VECTOR_2; /* Install CCB #0 interrupt vector */
```

```
/*IRQ level 2, enabled */
   brd->ch_ccb[3].vme_irq = INTR_ENABLED | LEVEL_2;
   printf ("\r\nDemonstrating 16-bit event counting...");
   printf ("\r\nEnter 16-bit Limit Count: ");
   getword (word_in);
   printf ("\r\n");
   brd->ch_ccb[3].cp.cp_w[0] = word_in;
   com2540 (event_16, 3);
   ch0_old = ch0_cnt;     /* using the ch0 interrupt svc routine */
   time = 0;
   com2540 (read_count, 3);

   while (!kbhit())    {
      if (resp_flg == count_ready) {
      word_in = brd->ch_ccb[3].cp.cp_w[1];
      printf ("\rChannel 0 event count: %.4X", word_in);

/*Check for the limit alarm each time count is updated. Alternatively
   the loop could be structure to check through each "while" iteration. */
     if (ch0_cnt > ch0_old) {
     printf ("\r\nLimit count interrupt!\r\n");

alrm_clr:
     brd->ch_ccb[3].cp.cp_b[8] = 0;
     ch0_old = ch0_cnt;  /* reset interrupt counter */
     time = 0;  /* reset timer */
     }
   else if (brd->ch_ccb[3].cp.cp_b[8] == limit_alarm) {
     printf ("\r\nLimit alarm detected!\r\n");
     goto alrm_clr;
     }
      com2540 (read_count, 3);
      time = 0;    /* reset timer */
      }
   else {
     printf ("\r<read_cnt> Command status = %.2X",
           resp_flg);
     time = 0;
     break;
     }
   if (time > 500) {
     printf ("\r\nTimeout waiting on event count!");
     break;
     }
     }
   prompt(anykey);
   if (byte_in == 'x') return;

/*Example of Period Measurement:
   the user is prompted for measurement clock select code, 16-bit hex
   average number, and then channel 2 is setup for period measurement.
   Since channel 1 is already generating a waveform, it may be convenient
   for the user to connect the output of channel 3 to the gate input of
   channel 2, GATE2+ at P3-B7 and GATE2- at P3-B8.
*/
   com2540 (disable_ch, 2); /* disable channel before next command */

/*user vector 3, IRQ level 3, enabled */
   brd->ch_ccb[2].vme_vec = USER_VECTOR_3; /* Install CCB ch2 intr vec */
   brd->ch_ccb[2].vme_irq = INTR_ENABLED | LEVEL_3;
   printf ("\r\nDemonstrating Period Measurement on Channel 2 ...");
   printf ("\r\nEnter clock select code [0..5]: ");
   getbyte (byte_in);
   per_clk = byte_in;
```

```
   brd->ch_ccb[2].gate_edge = per_clk;
   printf ("\r\nEnter sample size: ");
   getword (word_in);
   per_siz = word_in;
   brd->ch_ccb[2].cp.cp_w[0] = per_siz;
   ch2_old = ch2_cnt;
   com2540 (peri_meas_16, 2);
   printf("\r\nWaiting for initial measurement to be completed...");
   time = 0;
   while (!kbhit()) { /* loop which no key input */
     if (resp_flg != command_ack) {
         printf ("\r\nChannel status error = %d! ", ccb_status);
         break;

}

    if (ch2_cnt > ch2_old) { /* measurement complete interrupt */
        ccb_status = brd->ch_ccb[2].cp.cp_b[8];
    if (ccb_status != period_ready) {
        printf ("\r\nChannel status error = %d! ", ccb_status);
    break;
    }
    else {
        float_in = brd->ch_ccb[2].cp.cp_f[1];
        printf("\r\nMeasured period is %e", float_in);
    }
    com2540 (disable_ch, 2);
    brd->ch_ccb[2].vme_vec = USER_VECTOR_3; /* CCB ch2 intr vec */
    brd->ch_ccb[2].vme_irq = INTR_ENABLED | LEVEL_3;
    brd->ch_ccb[2].gate_edge = per_clk;
    brd->ch_ccb[2].cp.cp_w[0] = per_siz;
    com2540 (peri_meas_16, 2);
    time = 0;
    ch2_old = ch2_cnt;
    }
    if (time > 500)   {    /* Check time greater than 5 sec */
    printf ("\r\nTimeout waiting for period measurement!\r\n");
    break;
    }
    }
    prompt(anykey);
    if (byte_in == 'x')return;

/*Example of Quadrature Position Measurement:
  The clockwise signals from the incremental encoder are attached to
  CLK4+ at P3-A12 and CLK4- at P3-A13, while the CCW signals connect to
  CLK5+ at P3-A15 and CLK5- at P3-A16.
  The user is prompted for floating point scale (deg/pulse) value,
  the clockwise limit angle, the counter-clockwise limit angle,
  and then the channel pair (4,5) is setup for QPM.
*/

   qpc_flag = 0;
   prompt(qpm_option);
   if (byte_in == '0')   {
       qpc_flag = 1;
       for (i = 0; i < 4; i++)
            com2540 (disable_ch, i);
   }
   com2540 (disable_ch, 4); /* disable channel before next command */
   com2540( disable_ch, 5 );
   brd->ch_ccb[4].vme_vec = USER_VECTOR_4; /* Install CCB ch4 intr vec */

/* IRQ level 4, enabled */
```

```
    brd->ch_ccb[4].vme_irq = INTR_ENABLED | LEVEL_4;
    printf ("\r\nDemonstrating Quadrature Position Measurement"
            " on Channel 4 ...");
    printf ("\r\nEnter Float Scale value: ");
    getfp (float_in);
    brd->ch_ccb[4].cp.cp_f[0] = float_in;
    printf ("\r\nFloat CW Limit: ");
    getfp (float_in);
    brd->ch_ccb[4].cp.cp_f[1] = float_in;
    printf ("\r\nFloat CCW Limit: ");
    getfp (float_in);
    brd->ch_ccb[4].cp.cp_f[2] = float_in;
    com2540 (qpm, 4);
    ch4_old = ch4_cnt;
    if (qpc_flag) qpc_setup();
    printf ("\r\n");
    com2540 (read_qpm, 4);
    while (!kbhit()) {  /* do until key entered */
        if (ch4_old < ch4_cnt) {
          ch4_old = ch4_cnt;
          switch (brd->ch_ccb[4].gate_edge) {
             case 1 :
                printf ("\r\nClockwise 32-bit overflow!\r\n");
             break;
             case 2 :
                printf ("\r\nCounter-Clockwise 32-bit overflow!\r\n");
             break;
             case 3 :
                printf ("\r\nClockwise limit reached.\r\n");
             break;
             case 4 :
                printf ("\r\nCounter-Clockwise limit reached.\r\n");
             break;
          default: goto skpbreak;
          }
      }
      brd->ch_ccb[4].gate_edge = 0; /* acknowledge limit status */
skpbreak:
      if (resp_flg == qpm_ready) {
         float_in = brd->ch_ccb[5].cp.cp_f[0];
         printf ("\rChannel 4 Angular Position: %f", float_in);
      }
      else    {
         printf ("\r\n<read_qpm> Command status = %.2X", resp_flg);
         break;
      }
      com2540 (read_qpm, 4);
    }
    com2540 (disable_ch, 0);
    com2540 (disable_ch, 4);
    return;

}

void
qpc_setup(void)
{

/* If example set 1 not previously used, set default qpc clock select
    to 3 (20 usec period)        */
    com2540 (disable_ch, 0);
    if (scale == 0) scale = 3;
    brd->ch_ccb[0].gate_edge = scale;
    if (!qpc_clks) {
```

```
            printf ("\r\nEnter ch.0 qpc pulse width in clock pulses: ");
            getword (qpc_clks);
            brd->ch_ccb[0].cp.cp_w[0] = qpc_clks;
         }
         if (!qpc_delta) {
            printf ("\r\nEnter quadrature position delta: ");
            getword (qpc_delta);
            brd->ch_ccb[0].cp.cp_w[1] = qpc_delta;
         }
         com2540 (qpc, 0);      /* Quadrature Position Control on ch0 */
      }
      /* Timer interrupt service routine */
      #pragma interrupt()
      void
      time_isr (void)
      {
         time++;
      }
      /* Command status return interrupt service routine */
      #pragma interrupt()
      void
      resp_isr( void )
      {
            resp_flg = brd->response & 0x1F;
         if (resp_flg > qpm_ready) {
           req_err = 1;
            }
      }
      /* Channel measurement interrupt service routines */
      #pragma interrupt()
      void
      ch0_isr (void)
      {
         ch0_cnt++;
      }
      #pragma interrupt()
      void
      ch2_isr (void)
      {
         ch2_cnt++;
      }
      #pragma interrupt()
      void
      ch4_isr (void)
      {
         ch4_cnt++;
      }
      void
      com2540 (unsigned short command, unsigned char chan)
      {
         unsigned int tcmd, tcur, timeout;
         timeout = 5.0; /* set a 50 millisecond timeout */
         req_err = 0;
            resp_flg = 0xFF;
         brd->channel = chan;
         brd->cmd = command;
/* To allow program exit with interrupts inactive, check a null vector */
    if (brd->resp_vec == 0)return; /* return if no active vector */
    tcmd = time;          /* read initial time */
    while (resp_flg == 0xFF) {
```

```
      tcur = time;      /* read current time */
      if ((tcur - tcmd) > timeout) {
          printf ("\r\n<com2540> Reset command status timeout!!\r\n");
          while (!kbhit()); /* wait for user to hit a key */
          return;
      }
    }
    if (req_err) {
        printf ("\r\nCommand Status Error Code: ");
        switch (resp_flg) {
          case ch_alloc_err : printf ("Channel Allocation Error!\r\n");
          break;
          case bounds_err : printf ("Bounds Error!\r\n");
          break;
          case period_err : printf ("Period Error!\r\n");
          break;
          case pulse_err : printf ("Pulse Width Error!\r\n");
          break;
          case freq_err : printf ("Frequency Error!\r\n");
          break;
          case scale_err : printf ("Scale Error!\r\n");
          break;
          case qpm_err : printf ("QPM Error!\r\n");
          break;
          case gate_err : printf ("Gate Error!\r\n");
          break;
          case limit_err : printf ("Limit Error!\r\n");
          break;
          case ch_active_err : printf ("Channel Active Error!\r\n");
          break;
          case req_denied : printf ("Request Denied!\r\n");
          break;
        default: printf ("Command status =.2x", resp_flg);
        }
        while (!kbhit()); /* wait for user to hit a key */
    }
    return;
}
int
init2540 (void)
{
    unsigned char id, option;
time = 0;
ch0_cnt = 0;
ch0_old = 0;
ch2_cnt = 0;
ch2_old = 0;
ch4_cnt = 0;
ch4_old = 0;
printf ("\r\nEnter board address \(%.8X\): ", brd);
getlong ((unsigned int *) &brd);
id = (unsigned char )(brd->bid >> 8);
option = (unsigned char)(brd->bid & 0xff);
if (id != 0x25) {
  printf("\r\nNot a VMIVME-2540 ID @ %08x", brd);
  return (TRUE);
}
if (option < 1) {
  printf("\r\nVMIVME-2540 option will not support this program");
  return (TRUE);
}
```

```
    setvect (USER_VECTOR_0, &resp_isr );     /* install response ISR */
    brd->resp_vec = USER_VECTOR_0;           /* assign vector to IRQ */
    brd->resp_irq = 0x09;                    /* put response on IRQ 1 */
    com2540 (initialize, 0);                 /* 2540 asserts RESP_irq */
                                             /* when done */
                                             printf ("\r\nInitializing timer...");
    setvect (USER_VECTOR_1, &time_isr);      /* install timer ISR label */
    brd->tmr_ccb[1].vme_vec = USER_VECTOR_1;
          brd->tmr_ccb[1].vme_irq = 0x0A;    /* put timer on IRQ 2 level */
          brd->tmr_ccb[1].tp.tp_f[0] = 0.01; /* 100 Hz timer interrupt  */
          brd->channel = 1;                  /* point to timer 1     */
          brd->cmd = timer_16r               /* and start clock ticking */
    while (time < 2);   /* wait for first 2 ticks  */
    setvect (USER_VECTOR_2, &ch0_isr);   /* install ch0 ISR label */
    setvect (USER_VECTOR_3, &ch2_isr);   /* install ch2 ISR label */
    setvect (USER_VECTOR_4, &ch4_isr);   /* install ch4 ISR label */
    return(FALSE);
}
/****************************** END of gs.c ******************************/
```

The example program was compiled in a cross-compilation environment for a single-board processor (SPB) M68030 target. Certain functions are machine or OS dependent and may require modification to function properly in the user's environment. In particular, the function `kbhit()` returns a null if no key is entered and the character if a key is entered.

Timer 1 is used by the routine as the source of the polling interrupt. Thus, its ISR merely increments a variable. The command status ISR does more than just read the command status code, every error response is decoded fully and printed to the screen. It is recommended that the user program provides for error detection and processing for all command and channel status data acquisition software for the VMIVME-2540 since error detection decreases debug time. If the host CPU has an on-board timer interrupt source, it is not necessary to program the VMIVME-2540 for the timer 1 function.

The `com2540()` routine is the recommended algorithm for the VMIVME-2540 command / command status processing protocol. Both the reset command status and user command routines are bounded by the time variable, allowing no more than 5 seconds to elapse before reporting a timeout fault. Again, error detection decreases debug time.

The `init2540()` routine initializes the base address of the VMIVME-2540, then installs the response ISR and the timer 1 ISR. The command status interrupt uses VMEbus IRQ1, while the timer 1 interrupt uses IRQ2. The command status interrupt can be replaced with a polling function; however, obtaining the command status codes via the interrupt handler simplifies the process and maximizes efficiency for the VMIVME-2540 communication interface.

The main routine prompts the user for the selection of example set to be executed. Example_set_1 sets up the measurement functions for continuous measurement mode. For the purpose of this example, the most recent value of each measurement is merely read at a 100 Hz rate and printed to the screen. The continuous mode flags / channel data validity flags / measurement queue data ready flags were not used in the example, but generally one of the available flags should be used for detection of data measurement update by the local CPU. The integer measurement modes were chosen for this example to emphasize their speed and simplicity of programming.

Note that both the data and status are printed to the screen for integer period measurement on channels 6 and 7, data under the banner of 6_p and 7_p and status under the 6_s and 7_s banner. This arrangement allows the host CPU to continuously measure the period of an arbitrary waveform with high resolution over a broad range of time: from 400 ns to 13:1 ms with 200 ns resolution, and 13:1 ms to 1.31 s with 20 µs resolution. The host CPU always reads the four locations at each iteration, displaying the data from channels 6 and 7 with a status code of *period ready* or *scale error* after the delta angle of the QPC command has completed. Although the integer period measurement could have been programmed for autoranging using only one channel of the VMIVME-2540, the autoranging code requires a minimum of two input signal periods each time the clock frequency is changed when the autoranging is selected for the measurement. The example approach ensures the lowest latency between measurement updates for the host CPU.

This configuration for the VMIVME-2540 is usable for real-time motion control system with a position control loop update requirement of 100 Hz. In lieu of connecting the quadrature position output control signals to the integer QPM channels of the front panel connector, the signals could be connected to a motor for position and rate control. An incremental encoder on the elevator could provide both linear position measurement to the integer QPM channels and velocity feedback to the period measurement channels. By removing the *printf* statement from the main polling loop, the sampling frequency could easily be increased to 1,000 Hz.

Example_set_2 demonstrates the pulse train operation, frequency divider operation, event count operation, period measurement operation, and an addition example of quadrature measurement operation. This example set prompts the user for the parameter set associated with each of the operations, then performs a single measurement for each of the input operations. The example sets up a signal output on channel one (pulse train) from user-specified parameters, divides this signal by a user-specified divider on channel 1, inputs this divided signal on channel 3 for event counting with a user-specified limit, routes the channel 3 event count output signal to channel 2 for a period measurement. Note that this routing has the effect of providing a time duration for the event limit to occur. The example is not intended to represent any practical application, but to familiarize the user with the use of the product.

Appendix A presents the terminal output and a set of parameters using the above example program. Appendix A also contains a program which supports all CCB operations of the VMIVME-2540. No interrupts are used; this program relies completely on polling of CCB status and command status for all functions.

The VMIVME-2540 local CPU is a 68HC000 processor with no floating-point coprocessor, which adds many additional instruction cycles to each VMIVME-2540 measurement process for floating-point computations. This results in lower data throughput rates. The integer input operations should be used where feasible, and the floating-point math should be performed by the host CPU to achieve the highest possible data throughput rates using the VMIVME-2540.

Appendix

A

Example Code

Contents

Terminal Output of Program gs.c 118
Programming Example 120

Terminal Output of Program gs.c

```
Enter board address (FB000000): fb200000                              <-
Initializing timer...
Select example set to demonstrate (1 or 2): 1                         <-
Enter clock select code (1-5): 3                                      <-
Enter qpc pulse width in clock pulses: 200                            <-
Enter quadrature position delta: 2000
                                                                      <-
 <4,5>  <6p> <6s> <7p> <7s>
00000021 0FA0  03  0190 03                                      *
000007D0 FFFF  0E  FFFF 0E                                      *     <-

Press any key to continue (x to terminate sequence)...                <-
Select example set to demonstrate (1 or 2): 2                         <-
Demonstrating Pulse Train Generation on Channel 1 ...
Enter Float Period value: .001                                        <-
Enter Float Pulse Width: .0005                                        <-
Pulse train active on Ch. 1.

Press any key to continue (x to terminate sequence)...                <-
Demonstrating Frequency Division on Channel 0 ...
Enter 16-bit divider: 2                                               <-
Press any key to continue (x to terminate sequence)...                <-
Demonstrating 16-bit event counting...
Enter 16-bit Limit Count: 300                                         <-
Channel 0 event count: 0123
Limit count interrupt!
Channel 0 event count: 0129
Limit count interrupt!
Channel 0 event count: 001A                                           <-
Press any key to continue (x to terminate sequence)...                <-
Demonstrating Period Measurement on Channel 2 ...
Enter clock select code [0..5]: 3                                     <-
Enter sample size: 0                                                  <-
Waiting for initial measurement to be completed...
Measured period is 6.000000e-01
Measured period is 6.000000e-01
Measured period is 6.000000e-01                                       <-
Press any key to continue (x to terminate sequence)...                <-
Select QPM signal source internal (0) or external (1): 0              <-
Demonstrating Quadrature Position Measurement on Channel 4 ...
Enter Float Scale value: .0125                                        <-
Float CW Limit: 100.0                                                 <-
Float CCW Limit: 100.0                                                <-
Enter qpc pulse width in clock pulses: 200                            <-
Enter quadrature position delta: 20000                                <-
Channel 4 Angular Position: 99.812500                           *
```

```
Clockwise limit reached.
Channel 4 Angular Position: 250.05000                                  *   <-
Enter x to exit...any other key to re-select : x                           <-
Program terminating
<- Denotes a key entry including parametric entries with carriage return, space with
carriage return or carriage return.
*Denotes multi-reading display line.
```

Programming Example

```
/***  Name: Test2540.c  Demonstration Program for VMIVME-2540 board

    Description: This software was written to accommodate all the
           features of the 2540 product. To accomplish this
           objective, the program makes extensive use of
           pointers to all the data types and parameters
           in the CCB area of the shared memory space
           contained on the VMIVME-2540 board and uses
           many byte arrays to define data attributes. This
           approach allows any channel to be configured for
           any operation, with prompt provide to allow the
           function selection. This approach would not be
           necessary in a given application for the board,
           allowing a much simpler program to be used for
           dedicated channel operation. All pointers
           required for any operation are computed at the
           entry of the user-selected channel number, with
           the pointers computed as a function of the channel
           number for the subsequent operation. The program
           uses polling exclusively, and uses delay loop counts
           for paced access to the board.

    Notes: The program is code for a board base address of 0xFB200000.
       To accommodate another address, change BASE_ADDRESS prior to compilation.
 ***/

#define BASE_ADDRESS 0xFB200000
#define MAXACC  20
#define MAX_MWAIT 2000000000
#define MAX_CMD_RESP 200
#define MIN_DELAY 100  /* minimum delay of 50 usec */
#define NULL   0x00
#define FALSE  0
#define TRUE  -1
#define VAL_ID  0x25
#define BRD_OPT 0x03
#define DISABLE_CH 0x00
#define EVENT_16 0x01
#define EVENT_16_GATE 0x02
#define EVENT_32 0x03
#define EVENT_32_GATE 0x04
#define READ_CURR_EVT_CNT 0x06
#define FREQ_DIV_16 0x07
#define FREQ_DIV_32 0x08
#define SQ_WAVE 0x09
#define PULS_TRAIN 0x0A
#define FREQ_GEN 0x0B
```

```
#define DUTY_CYCLE 0x0C
#define MEAS_FLT_PER 0x0D
#define MEAS_FLT_PER_ENH 0x0E
#define MEAS_FLT_FREQ 0x0F
#define MEAS_FLT_FREQ_ENH 0x10
#define MEAS_PULS_WIDTH_FLT 0x11
#define MEAS_PULS_WIDTH_FLT_ENH 0x12
#define MEAS_QUAD_POS_FLT 0x13
#define READ_QUAD_POS_FLT 0x15
#define MEAS_QUAD_POS_INT 0x16
#define READ_QUAD_POS_INT 0x17
#define TIMER_INTERRUPT_SNG 0x18
#define TIMER_INTERRUPT_MULT 0x19
#define DISABLE_TIMER 0x1A
#define INITIALIZE 0x1B
#define RESET_STATUS 0x1C
#define BLOCK_MOVE 0x1D
#define EXECUTE 0x1E
#define ECHO_PC 0x1F
#define MEAS_PER16 0x20
#define PULS_SEQ 0x21
#define PGM_IO 0x22
#define QUAD_POS_CTL 0x23
#define GROUP_ACQ 0x24
#define EVENT_DELAY_TRIG 0x25
#define MEAS_PULS_WIDTH_INT 0x26
#define MEAS_PER32 0x27
#define MEAS_PUL32 0x28
#define ACT_HI  0x00
#define ACT_LO  0x20
#define MEAS_RDY 0xFF
#define TFLAG 0xFF
#define ACK 0x01
#define EVT_RDY 0x02
#define PER_RDY 0x03
#define FRQ_RDY 0x04
#define PUL_RDY 0x05
#define QPM_RDY 0x06
#define LIM_ALARM 0x07
#define INT_32 1
#define INT_16 2
#define FLT_32 3
#define SINT_32 4
#define DUAL_SINT_32 5
#define INPUT 0
#define OUTPUT 0xFF
#define PRE_SCALE 0xFF
#define RESTART -1
#define EXIT -2
#define NUM_INPUTS 18
```

```
#define SNGL 0
#define DUAL 1

/* Define flags for input operation mode */
#define CCB_STAT 0x01
#define MFLAG 0x02
#define MQUEUE 0x04
#define CONTINUOUS 0x80      /* opn_flag: continuous mode indicator */
#define OPN_COMPLETE 0xFF
#define DISCRETE 0x10       /* opn_flag: discrete mode */
#define SET_UP 0x08        /* opn_flag: setup only */

/* Define the configuration channel size for each option */
char *nchptr[] = { "4", "8", "16", "24" };

/* Define the command status and CCB status codes indexed by code */
char *diag_msg[] = { "", "Command Acknowledge", "Event count ready",
        "Period Measurement ready","Freq. Measurement ready",
        "Pul. Wdth. Measurement ready", "Quad. pos. measurement ready",
        "Limit Alarm", "Timer Alarm", "Channel Allocation error",
        "Bounds error", "Period error", "Pulse Width error",
        "Freq. error", "Scale error", "Quad. Pos. meas. error",
        "Gate error", "Limit error", "Active channel error",
        "Request Denied", "Under-range measurement",
        "QPM clockwise overflow", "QPM counter-clockwise overflow" };

/* Define a message timeout error code string */
char timer_str[] = "Measurement flag timeout\r\n";

/* Message for any code not in above list */
char unk_str[] = "Unknown return code\r\n";

/* Define the quadrature messages codes */
char *quad_diag[] = { "", "Clockwise 32-bit Overflow", "Counterclockwise"
           " 32-bit overflow", "Clockwise Limit Exceeded",
           " Counterclockwise limit exceeded" };

/* Define any fatal/non-fatal attribute for each error code */
char fatal_flg[] = { FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
          FALSE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
          TRUE, TRUE, TRUE, FALSE };
char chnl_active[24] = { FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
    FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
    FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE };

/* define an array of address (1 for each channel) which point to the meas.
  location in the CCB. */
unsigned char *chnl_meas_ptr[24] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };
```

```
/* define an array of addresses (1 per channel) which point to CCB status */
unsigned char *chnl_status_ptr[24] = { NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };

/* define the commands by program index */
unsigned char cmd_sel[27] = { MEAS_PER32, MEAS_PUL32, MEAS_PER16, MEAS_FLT_PER,
     MEAS_FLT_PER_ENH, MEAS_FLT_FREQ, MEAS_FLT_FREQ_ENH, EVENT_16,
     EVENT_32, MEAS_PULS_WIDTH_FLT, MEAS_PULS_WIDTH_FLT_ENH,
     MEAS_QUAD_POS_FLT, MEAS_QUAD_POS_INT, GROUP_ACQ,READ_CURR_EVT_CNT,
     READ_QUAD_POS_FLT,READ_QUAD_POS_INT, EVENT_DELAY_TRIG, SQ_WAVE,
     PULS_TRAIN, FREQ_GEN, DUTY_CYCLE, PULS_SEQ, FREQ_DIV_16,
     FREQ_DIV_32, PGM_IO, QUAD_POS_CTL };
unsigned char cmd_index_tbl[41] = { FALSE, 7, 7, 8, 8, FALSE, 14, FALSE,
     FALSE, FALSE, FALSE, FALSE, FALSE, 3, 4, 5, 6, 9, 10, 11, FALSE,
     FALSE, 12, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE,
     FALSE, 2, FALSE, FALSE, FALSE, 13, 17, FALSE, 0, 1 };

/* define the data types for screen display */
char opn_data_types[27] = { INT_32, INT_32, INT_16, FLT_32, FLT_32, FLT_32,
            FLT_32, INT_16, INT_32, FLT_32, FLT_32, FLT_32,
            SINT_32, DUAL_SINT_32, FLT_32, FLT_32, SINT_32,
            FLT_32, FLT_32, FLT_32, FLT_32, FLT_32, FLT_32,
            FLT_32, FLT_32, FLT_32, SINT_32 };

/* flag the command types needing a pre-scalar */
char prescl_flg[27] = { NULL, NULL, NULL, NULL, PRE_SCALE, NULL, PRE_SCALE,
    NULL, NULL, NULL, PRE_SCALE, NULL, NULL, NULL, NULL, NULL, NULL,
    NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL };

/* flag the command types that use period measurement floating point */
char flt_per_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
             NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
             NULL, NULL, FLT_32, FLT_32, NULL, NULL, NULL, NULL,
             NULL, NULL, NULL };
/* flag the command types that use frequency measurement floating point */
char flt_frq_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
             NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
             NULL, NULL, NULL, NULL, FLT_32, FLT_32, NULL, NULL,
             NULL, NULL, NULL };

/* flag the command types that use pulse width measurement floating point */
char flt_pw_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, FLT_32, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL };

/* flag the command types that use 16-bit and 32-bit frequency division */
char freq_div_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
```

```
             NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
             NULL, NULL, NULL, NULL, NULL, NULL, NULL, INT_16,
             INT_32, NULL, NULL };
/* flag the command types that use gate input signals */
char gate_flg[27] = { NULL, TFLAG, NULL, NULL, NULL, NULL, NULL, TFLAG,
           TFLAG, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL, NULL, NULL };
/* flag the command types that use clock selects */
char clock_flg[27] = { TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, NULL,
            NULL, TFLAG, TFLAG, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, TFLAG, NULL, NULL,
            NULL, TFLAG };
/* flag the command types that use limit parameters */
char lim_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, TFLAG,
           TFLAG, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL };
/* flag the command types that use a duty_cycle parameter */
char duty_cyc_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
          NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
          NULL, NULL, NULL, NULL, TFLAG, NULL, NULL, NULL, NULL,
          NULL };
/* flag the command types that use clocks counts high and low for signal def */

char clock_sel_flg[27] = { NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
           NULL, NULL, NULL, NULL, NULL, TFLAG, NULL, NULL, NULL,
           NULL };
/* flag the command types that use sample sizes */
char sampl_flg[27] = { NULL, NULL, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, NULL,
            NULL, TFLAG, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL };
/* flag the command types where continuous/discrete is applicable */
char cont_flg[27] = { TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG,
            TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG, TFLAG,
            TFLAG, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
            NULL, NULL };

/* define the measure status expected for normal measurement completion */
unsigned char stat_sel[27] = { PER_RDY, PUL_RDY, PER_RDY, PER_RDY, PER_RDY,
                FRQ_RDY, FRQ_RDY, EVT_RDY, EVT_RDY, PUL_RDY,
                PUL_RDY, QPM_RDY, QPM_RDY, QPM_RDY, EVT_RDY,
                QPM_RDY, QPM_RDY, LIM_ALARM, NULL, NULL, NULL,
                NULL, NULL, NULL, NULL, OPN_COMPLETE };

/* Define all prompt message strings */
char sel_period[] = "\r\nSelect the period desired (Sec. - xxx.yy): >";
char sel_freq[] = "\r\nSelect the frequency desired (Hz. - xxx.yy): >";
```

```
char sel_puls_width[] = "\r\nSelect the pulse width desired (Sec. - xxx.yy): >";
char sel_duty_cycle[] = "\r\nSelect the duty cycle desired (\% - xxx.yy): >";
char sel_high_clocks[] = "\r\nSelect high level duration"
    " (number of clocks): >";
char sel_low_clocks[] = "\r\nSelect low level duration"
    " (number of clocks): >";
char sel_total_dur[] = "\r\nSelect total pulse sequence duration (2-32767) : >";
char div_prm[] = "\r\nEnter Divisor: >";
char opn_type[] = "\r\n\nSelect operation type\r\n"
   " (0 = INPUT; 1 = OUTPUT; 2 = DISABLE): >";
char meas_type[] = "\r\n\nSelect Measurement Type\r\n"
 " 0 = 32-bit Integer Period          1 = 32-bit Integer Pulse Width\r\n"
 " 2 = 16-bit Integer Period          3 = Floating Period\r\n"
 " 4 = Enhanced Floating Period       5 = Floating Frequency\r\n"
 " 6 = Enhanced Floating Frequency    7 = 16-bit Integer Event Count\r\n"
 " 8 = 32-bit Integer Event Count     9 = Floating Pulse Width\r\n"
 " 10 = Enhanced Floating Pulse       11 = Quad Pos. Floating Measurement\r\n"
 " 12 = Quad Pos. Integer Measuremnt        13 = Quad Group Acquisition\r\n"
 " 14 = Read Current Event Count      15 = Read Quad Position Float\r\n"
 " 16 = Read Quad Position Int.       17 = Event Delay Trigger\r\n"
 ">";
char output_type[] = "\r\nSelect Output Operation\r\n"
 " 0 = Square Wave (50/50)     1 = Pulse Train\r\n"
 " 2 = Frequency  (50/50)      3 = Duty Cycle\r\n"
 " 4 = Pulse Sequence          5 = 16 bit Frequency Divider\r\n"
 " 6 = 32-bit Frequency Divider       7 = Programmed Output\r\n"
 " 8 = Quadrature Position Control\r\n"
 ">";
char qpc_pw_sel[] = "\r\nSelect the output pulse width (as the number of\r\n"
         "clock source cycles): >";
char qpc_delta_sel[] = "\r\nSelect the delta position (in output pulses): >";
char qpc_pos_sel[] = "\r\nSelect the current position location (in output\r\n"
          " cycles): >";
char wait_outmsg[] = "\r\nWait for the angle change to complete (y or n): >";
char opn_prm[] = "\r\n\nSelect input operation mode\r\n"
       "(0 = single; 1 = continuous; 2 = setup only; "
       "3 continuous-setup): >";
char chnnum[] = "\r\n\nEnter Channel number: >";
char sel_pulse[] = "\r\nSelect active gate level/active clock edge\r\n"
 "(0 = active high gate/rising clock; 1 = active high gate/falling clock;\r\n"
 " 2 = active low gate/rising clock; 3 = active high gate/falling clock):\r\n"
 " 4 = no active gating: >";
char sel_clock[] = "\r\n"
  "Select clock frequency desired (0 = autoranging; 1 = 200 nsec.;\r\n"
  "2 = 2 usec.; 3 = 20 usec; 4 = 200 usec.; 5 = 2 msec.): >";
char sel_limit[] = "\r\nSelect limit for the limit alarm: >";
char sel_level[] = "\r\nSelect output level (0 = low; 1 = high): >";
char mode_prm[] = "\r\n\nEnter mode (0 = report every measurement;"
        " 1 = average \"N\" measurement;"
        "\r\n 2 = discard \"N\" measurements) : >";
```

```
char sampl_prm[] = "\r\nEnter sample size (0-65535): >";
char prescale[] = "\r\nEnter a prescaler choice (0-65535): >\r\n";
char pang_lim_sel[] = "\r\nEnter clockwise angle limit: >";
char nang_lim_sel[] = "\r\nEnter counter-clockwise angle limit: >";
char scale_sel[] = "\r\nEnter a scale facter (degrees/pulse): >";
char sel_clock_edge[] = "\r\nSelect the desired input edge\r\n"
    "(0 = falling; 1 = rising): >";
char spres_flag[] = "\r\nRetriggering suppressed (y or n)? >";
char sel_delay_time[] = "\r\nEnter the desired delay from the input edge\r\n"
    "in seconds (xx.yyyyy): >";
char status_poll[] = "\r\n"
        "Poll the CCB status (1), Measure Ready flag (2) or\r\n"
        "Measurement Queue flag (3): >";
char display_meas[] = "\r\n"
        "Display the measurement data (y/n)";
char restart_opt[] = "\r\nAutomatic restart (y or n): >";
char cont_meas[] = "\r\nContinue operations on active channel (y or n)? >";
char operation_sel[] = "\r\nEnter:\r\n"
   "\"r\" to disable;\r\n"
   "\"a\" to select another channel with current channel active;\r\n"
   "\"c\" to continue with current channel measurements;\r\n"
   "\"x\" to exit;\r\n"
   "any other key for disable/enable restart sequence...\r\n"
   "> ";
char continue_msg[] = "\r\nEnter \"y\" to continue from current position,"
           "\"n\" to re-initialize: >";
char anykey[] = "\r\nPress any key to continue\r\n";
char invalid_sel[] = "\r\nInvalid selection - Re-enter\r\n";
char inv_lvl[] = "\r\nInvalid entry for this operation - Re-enter\r\n";
char inv_brd_id[] = "\r\nBoard installed has incorrect ID code.\r\n";

struct vmivme_2540_regs {
    unsigned short bid; unsigned short rev; unsigned short cmd;
    unsigned char unused; unsigned char response; unsigned char resp_irq;
    unsigned char resp_vec; unsigned char channel; unsigned char contdisc;
    unsigned short mflag; unsigned char mchan; unsigned char mcode;
  struct CH_CB {
    unsigned char command; unsigned char gate_edge; unsigned char vme_irq;
    unsigned char vme_vec;
  union {
    unsigned char cp_b[12]; unsigned short cp_w[6]; unsigned int cp_l[3];
    float cp_f[3];
       } cp;
   } ch_ccb[24];
   struct TMR_CB {
    unsigned char tcmd; unsigned char tgate; unsigned char vme_irq;
    unsigned char vme_vec;
   union  {
    unsigned char tp_ub[12]; char tp_b[12]; unsigned short tp_uw[6];
    short tp_w[6]; unsigned int tp_ul[3]; int tp_l[3];
```

```
    float tp_f[3];
    } tp;
   } tmr_ccb[6];
    unsigned char cd[24];   unsigned char valid_flgs[24];
    unsigned char reserv[480];    unsigned short diag_bufr[32];
} *brd_ptr = (struct vmivme_2540_regs *)BASE_ADDRESS;

char wait_for_resp(), wait_for_meas(), poll_resp_code();
int prompti();
float promptf();
void wait_for_valid(), print_meas(), access_delay();

short ncounts;
static int ccb_length = 16;
unsigned char meas_st, *meas_addr, poll_flag, active_cmd, meas_cmd;

unsigned char *mstatus, *cvalue, *valid_flg, *brd_id_ptr,
       *clk_value, *con_disc, *curr_cmd, *curr_op_mode;

unsigned char clock_sel, data_typ, data_dir, chnl_cmd, read_opn, clk_edge,
       restart, opn_flag, sflag;

char dis_flag;

unsigned short *diagnostics, *command, *svalue, *freq_div;

/* Allocate pointers to access the various parameter types */
unsigned char *uc_ptr0, *uc_ptr1, *uc_ptr2, *uc_ptr4, *uc_ptr5,
       *uc_ptr8, *uc_ptr10;
short *s_ptr0, *s_ptr1;
unsigned short *us_ptr0, *us_ptr1, *us_ptr2, *us_ptr5;
unsigned int *ul_ptr0, *ul_ptr1;
float *flt_ptr0, *flt_ptr1, *flt_ptr2;

/* allocate variables for entry by user */
short qpc_delta;
unsigned short qpc_puls_cnt;
int qpc_position, *slvalue, *slvalue1;
float freq_sel, puls_sel, pos_ang_lim, neg_ang_lim, ang_scale, tdelay;

/* Allocate pointers to selected channels */
struct CH_CB *brd_chnl, *brd_chnl1;

void
main()
{
    static char opt_chnls[4] = { 4, 8, 16, 24 };
    int ntime, cmd_index, act_pulse, act_chnl, in_samp, op_type,
      op_mode, pscale_clk, max_chnls;
    unsigned char brd_id, out_level, m_mode, res_flag;
```

```
    short cmd_wait, i;
    unsigned short clk_low, clk_high, clk_dur;
    unsigned long f_divisor, limit;
    char in_char, ch_opt, div_flag, cmd_status, wait_out, quad_con;
    unsigned int *ev_32_lim_ptr;

    command = (unsigned short *)&brd_ptr->cmd;
    diagnostics = (unsigned short *)&brd_ptr->diag_bufr[0];

begin:

/* Display board menu */
sel_chn:
    res_flag = FALSE;
    act_chnl = prompti(chnnum);   /* prompt for channel number */
    if (act_chnl == EXIT)return;

/* Now access the board, retrieving the ID and option and allow enough
  time for board to complete initialization (if it was reset/power-on). */
    brd_id_ptr = (unsigned char *)&brd_ptr->bid;
    while (*brd_id_ptr != VAL_ID)   {
      ntime--;
      access_delay();
      if (ntime <= 0) {
        fast_print(inv_brd_id);
        return;    }
    }
    con_disc = &brd_ptr->contdisc;  /* set continuous discrete flag ptr */
    brd_ptr->channel = act_chnl;   /* set pointer to selected channel */
    ch_opt = *(brd_id_ptr+1);    /* retrieve board option */
    max_chnls = (int )opt_chnls[ch_opt];  /* retrieve number of channels */
    printf("\r\nNumber of channels = %2d", max_chnls);
    if (act_chnl > max_chnls)    {   /* check selected channel */
      fast_print(invalid_sel);
      goto sel_chn;        }

/* First send the reset command status code */
    *command = (unsigned short )RESET_STATUS;
    if (wait_for_resp())goto sel_chn;

/* set the active channel number in the header shared memory space */
    brd_ptr->channel = act_chnl;

/* set default non-read operation flag */
    read_opn = FALSE;

/* set default zero length pulse width */
    act_pulse = 0;
    valid_flg = &brd_ptr->valid_flgs[act_chnl];
    brd_chnl = &brd_ptr->ch_ccb[act_chnl];
```

```
    brd_chnl1 = &brd_ptr->ch_ccb[act_chnl+1];
    curr_cmd = (unsigned char *)&brd_chnl->command;
    cvalue = (unsigned char *)&brd_chnl->gate_edge;
    uc_ptr0 = &brd_chnl->cp.cp_b[0];
    uc_ptr1 = &brd_chnl->cp.cp_b[1];
    uc_ptr2 = &brd_chnl->cp.cp_b[2];
    uc_ptr4 = &brd_chnl->cp.cp_b[4];
    uc_ptr5 = &brd_chnl->cp.cp_b[5];
    uc_ptr8 = &brd_chnl->cp.cp_b[8];
    uc_ptr10 = &brd_chnl->cp.cp_b[10];
    flt_ptr0 = (float *)uc_ptr0;
    flt_ptr1 = (float *)uc_ptr4;
    flt_ptr2 = (float *)uc_ptr8;
    ul_ptr0 = (unsigned int *)uc_ptr0;
    ul_ptr1 = (unsigned int *)uc_ptr4;
    us_ptr0 = (unsigned short *)uc_ptr0;
    us_ptr1 = (unsigned short *)uc_ptr2;
    us_ptr2 = (unsigned short *)uc_ptr4;
    us_ptr5 = (unsigned short *)uc_ptr10;
    s_ptr0 = (short *)us_ptr0;
    s_ptr1 = (short *)us_ptr1;
    clk_value = uc_ptr1;
    meas_addr = uc_ptr4;
    slvalue = (int *)uc_ptr4;
    svalue = us_ptr1;
    active_cmd = *curr_cmd;
    curr_op_mode = &brd_ptr->cd[act_chnl];
    if (active_cmd != 0)
      if (prompt(cont_meas) == 'y') {
        meas_cmd = active_cmd;
        cmd_index = cmd_index_tbl[active_cmd];
        if (cmd_index == FALSE) {
          printf("\r\nCan't continue this channel.\r\n");
          res_flag = FALSE;
          goto sel_chn;    }
        res_flag = TRUE;
        opn_flag = chnl_active[act_chnl];
        data_typ = opn_data_types[cmd_index];
        mstatus = chnl_status_ptr[act_chnl];
        meas_st = stat_sel[cmd_index];
        meas_addr = chnl_meas_ptr[act_chnl];
        data_dir = INPUT;
        op_type = INPUT;
        if (opn_flag & CONTINUOUS)goto cloop;
        else goto resume_m;  /* discrete channel operations */
      }
sel_opn:
    op_type = prompti(opn_type);   /* select operation type */
    if (op_type == EXIT)return;
    if (op_type == RESTART)goto sel_chn;
```

```
    if (op_type == 2) {
       *command = (unsigned short )DISABLE_CH;
       wait_for_resp();
       goto sel_chn;
    }
    if (op_type == 0)   {
      data_dir = INPUT;
      cmd_index = prompti(meas_type);
    }
    else
      if (op_type == 1) {
        data_dir = OUTPUT;
        opn_flag = DISCRETE;
        cmd_index = prompti(output_type);
        cmd_index = cmd_index + NUM_INPUTS;
      }
    else {
      fast_print(invalid_sel);
      goto sel_opn;
    }
    div_flag = freq_div_flg[cmd_index];
    if (cmd_index == EXIT)return;

/* Continue to process menu selection */
    data_typ = opn_data_types[cmd_index];
    meas_cmd = cmd_sel[cmd_index];
    meas_st = stat_sel[cmd_index];
    mstatus = uc_ptr8;
    if (meas_cmd == MEAS_QUAD_POS_FLT)   {
      pos_ang_lim = promptf(pang_lim_sel);
      neg_ang_lim = promptf(nang_lim_sel);
      ang_scale = promptf(scale_sel);
      meas_addr = uc_ptr0;
    }
    if (meas_cmd == EVENT_DELAY_TRIG)    {
      clk_edge = prompti(sel_clock_edge);
      tdelay = promptf(sel_delay_time);
      sflag = prompt(spres_flag);
      if (sflag == 'y')
        sflag = 0xFF;
      else sflag = 0;
      if (prompt(restart_opt) == 'y') restart = TRUE;
      else restart = FALSE;
    }
    if (clock_flg[cmd_index]) {
re_clk:
      clock_sel = (unsigned char)prompti(sel_clock);
      if ((clock_sel < 0) || (clock_sel > 5)) {
        fast_print(invalid_sel);
        goto re_clk;            }
```

```
    }
    if (prescl_flg[cmd_index] == (char )PRE_SCALE)
      pscale_clk = prompti(prescale);
    else pscale_clk = 0;
    if (gate_flg[cmd_index])    {
      act_pulse = prompti(sel_pulse); /* prompt for act. low/high */
      if (act_pulse != 4) {
        if (meas_cmd < READ_CURR_EVT_CNT) meas_cmd += 1; }
      else
        act_pulse = 0;
    }
    if (lim_flg[cmd_index])   {
re_lim: limit = prompti(sel_limit);
      if (meas_cmd < EVENT_32)
        if (limit > 65535) {
          fast_print(inv_lvl);
          goto re_lim;  }
    }
    if (cmd_index < 2)  {
re_m_mode:
      m_mode = (unsigned char )prompti(mode_prm);
      if (m_mode > 2)   {
        fast_print(invalid_sel);
        goto re_m_mode; }
      if (m_mode != 0) sampl_flg[cmd_index] = TFLAG;
      else sampl_flg[cmd_index] = NULL;
    }
    if (sampl_flg[cmd_index])
      in_samp = prompti(sampl_prm);
    if (flt_per_flg[cmd_index])
      freq_sel = promptf(sel_period);
    if (flt_frq_flg[cmd_index])
      freq_sel = promptf(sel_freq);
    if (flt_pw_flg[cmd_index])
      puls_sel = promptf(sel_puls_width);
    if (duty_cyc_flg[cmd_index])
      puls_sel = promptf(sel_duty_cycle);
    if (clock_sel_flg[cmd_index])  {
      clk_low = prompti(sel_low_clocks);
      clk_high = prompti(sel_high_clocks);
      clk_dur = prompti(sel_total_dur);
    }
    if (div_flag)  {
re_div: f_divisor = prompti(div_prm);
      if (f_divisor < 2) goto div_err;
      else
        if (f_divisor > 65535)
          if (div_flag == INT_16) {
div_err:       fast_print(inv_lvl);
            goto re_div;     }
```

```
    }
    if (meas_cmd == PGM_IO)  {
re_lvl: out_level = (unsigned char )prompti(sel_level);
      if ((out_level < 0) || (out_level > 1)) {
        fast_print(invalid_sel);
        goto re_lvl;            }
    }
    else
      if (meas_cmd == MEAS_PER16)
        mstatus = uc_ptr2;
    else
      if (meas_cmd == QUAD_POS_CTL)  {
qloop:
        qpc_delta = (short )prompti(qpc_delta_sel);
        if (*curr_cmd == QUAD_POS_CTL)
          if (prompt(continue_msg) == 'y') {
            quad_con = TRUE;
            goto del_only;
          }
        quad_con = FALSE;
        qpc_puls_cnt = (unsigned short )prompti(qpc_pw_sel);
        qpc_position = (int )prompti(qpc_pos_sel);
del_only:
        printf("\r\ndelta angle = %d", qpc_delta);
        wait_out = prompt(wait_outmsg);
        mstatus = uc_ptr10;
      }

resume_m:
    if (cont_flg[cmd_index])    {
      op_mode = prompti(opn_prm);  /* prompt for discrete/continuous */
      if (op_mode == 0) opn_flag = DISCRETE;
      else if (op_mode == 1)opn_flag = CONTINUOUS;
        else if (op_mode == 2)opn_flag = SET_UP;
          else opn_flag = CONTINUOUS | SET_UP;
    }
    if (data_dir == INPUT) {
      poll_flag = (unsigned char)prompti(status_poll);
      dis_flag = prompt(display_meas);
      if (dis_flag != 'y')dis_flag = NULL;
    }
    poll_flag &= 0x03;
    if (poll_flag == 0x03)poll_flag++;
    if (opn_flag & CONTINUOUS) opn_flag |= poll_flag;
    chnl_active[act_chnl] = opn_flag;

/* Set pointers to diagnostic buffer and board and
  clear the diagnostics buffer */
    diagnostics = (unsigned short *)&brd_ptr->diag_bufr[0];
    for (i = 0; i < 12; i++) *diagnostics++ = 0;
```

```
    ntime = MAXACC;
    *command = RESET_STATUS;
    if (wait_for_resp()) goto sel_chn;
    in_char = NULL;
mloop:
    if (res_flag)  {
      res_flag = FALSE;
      goto send_cmd;
    }
    if (meas_cmd < FREQ_DIV_16) {
      *cvalue = act_pulse;
      if (meas_cmd < EVENT_32) {
        meas_addr = (unsigned char *)uc_ptr2;
        *us_ptr0 = (unsigned short)limit;

      }
      else
        if (meas_cmd < READ_CURR_EVT_CNT)  {
          if (clock_sel == 4) clock_sel = 0;
          *ul_ptr0 = limit;
        }
        else  { /* Read current event count */
          read_opn = TRUE;
          chnl_cmd = *curr_cmd;
          if (chnl_cmd < EVENT_32)
            data_typ = INT_16;
          else
            data_typ = INT_32;
          goto send_cmd; /* command is a read event count */
        }
    }
    else
      if (meas_cmd < FREQ_DIV_32)
        *us_ptr0 = (unsigned short )f_divisor;
    else
      if (meas_cmd < SQ_WAVE)
        *ul_ptr0 = f_divisor;
    else
      if ((meas_cmd < MEAS_FLT_PER)) {
        *flt_ptr0 = freq_sel;
        *flt_ptr1 = puls_sel;
      }
    else
      if (meas_cmd < MEAS_QUAD_POS_FLT)      {
        *cvalue = (unsigned char )clock_sel;
        *us_ptr5 = (unsigned short )pscale_clk;
        *us_ptr0 = (unsigned short )in_samp;
      }
    else
      if (meas_cmd < MEAS_QUAD_POS_INT) {
```

```
      mstatus = cvalue;
      if (meas_cmd == READ_QUAD_POS_FLT)   {
        read_opn = TRUE;
        goto send_cmd;           }
      else              {
        *flt_ptr0 = ang_scale;
        *flt_ptr1 = pos_ang_lim;
        *flt_ptr2 = neg_ang_lim;
      }
    }
  else
    if (meas_cmd < TIMER_INTERRUPT_SNG) {
      mstatus = cvalue;
      meas_addr = uc_ptr0;
      if (meas_cmd == READ_QUAD_POS_INT) read_opn = TRUE;
    }
  else
    if (meas_cmd < PULS_SEQ) {
      *cvalue = (unsigned char )clock_sel;
      meas_addr = uc_ptr0;
    }
  else
    if (meas_cmd < PGM_IO)   {
      *cvalue = (unsigned char)clock_sel;
      *us_ptr0 = clk_low;
      *us_ptr1 = clk_high;
      *us_ptr2 = clk_dur;
    }
  else
    if (meas_cmd < QUAD_POS_CTL)
      *cvalue = out_level;
  else
    if (meas_cmd < GROUP_ACQ) {
      *cvalue = (unsigned char)clock_sel;
      *s_ptr1 = qpc_delta;
      if (quad_con) goto send_cmd;
      else  {
        *ul_ptr1 = qpc_position;
        *us_ptr0 = qpc_puls_cnt;
      }
    }
  else
    if (meas_cmd < EVENT_DELAY_TRIG)  {
      mstatus = cvalue;
      slvalue = (int *)ul_ptr0;
      slvalue1 = (int *)((int )slvalue + (2 * ccb_length));
      read_opn = TRUE;
    }
  else
    if (meas_cmd < MEAS_PULS_WIDTH_INT) {
```

```
        *cvalue = clk_edge;
        *flt_ptr0 = tdelay;
        *uc_ptr4 = restart;
        *uc_ptr5 = sflag;
      }
    else
      if (meas_cmd > MEAS_PULS_WIDTH_INT)  {
        *cvalue = act_pulse; /* set the active pulse level */
        *uc_ptr0 = m_mode;   /* set msnmnt mode */
        *clk_value = clock_sel; /* select the active clock */
        *us_ptr1 = (unsigned short )in_samp; /* set sample size */
      }
    else
      *cvalue = 0;
send_cmd:
    chnl_meas_ptr[act_chnl] = meas_addr;
    chnl_status_ptr[act_chnl] = mstatus;
    if (opn_flag & CONTINUOUS)
      *con_disc = 0xff;
    else
      *con_disc = 0;
    *valid_flg = 0;
    *mstatus = 0;       /* clear status */
    *command = meas_cmd;
    if (wait_for_resp())goto sel_chn;
    if (data_dir == OUTPUT) {
      if (meas_cmd != QUAD_POS_CTL)
        goto sel_chn;
      else
        if (wait_out == 'n')goto sel_chn;
    }
cloop:
    if (op_type == 0)
      if (opn_flag & SET_UP) {
        printf("\r\nInput setup complete\r\n");
        chnl_active[act_chnl] = opn_flag ^ SET_UP;
        goto sel_chn;
      }
    if (opn_flag & 0x01);
    else
      if (opn_flag & 0x02)  {
        mstatus = valid_flg;
        meas_st = 0xff;
      }
      else
        if (opn_flag & 0x04) {
          mstatus = (unsigned char *)&brd_ptr->mflag;
          meas_st = 0xff;
        }
    if (meas_st == NULL) goto begin;
```

```
    else {
      if (opn_flag & CONTINUOUS)
        while (!kbhit())   {
          cmd_status = wait_for_meas();
          if (cmd_status) break;
          if (read_opn) *command = meas_cmd;
        }
      else wait_for_meas();
      brd_ptr->channel = act_chnl;

/* Channel disable - send selected command */
      in_char = prompt(operation_sel);
      if (in_char == 'x')return;
      else
        if (in_char == 'c')
          if (*curr_cmd == QUAD_POS_CTL)goto qloop;
          else if (data_dir == INPUT)goto cloop;
          else goto sel_chn;
      else
        if (in_char == 'a') goto sel_chn;
      *command = (unsigned short )DISABLE_CH;
      if (wait_for_resp())goto sel_chn;
      if (in_char == 'r')goto begin;
      if (read_opn)goto sel_chn;
      else goto mloop;
    }
}

/* Define the function which waits for a command status and processes
  the status received from the polling routine. */
char
wait_for_resp(void)
{
    short cmd_wait, i, j;
    char rtn_code, cmd_out;

    cmd_wait = 0;
    rtn_code = 0;
    cmd_out = *command;
    while (rtn_code == 0)          {
      rtn_code = poll_resp_code();
      if (cmd_out == RESET_STATUS)
        if (rtn_code == 0) return(FALSE);
        else rtn_code = 0;
      cmd_wait++;
      if (cmd_wait > MAX_CMD_RESP) {
        printf("\r\n***No response for command***\r\n");
        return(TRUE);
      }
    }
```

```
    if (*diagnostics != 0)   {
      printf("\r\nCommand fatal error - diagnostic buffer:\r\n");
      for (i = 0; i < 7; i++)   {
        for (j = 0; j < 6; j++)
           printf("0x%04x ",*diagnostics++);
        printf("\r\n");
      }
    }
    if (read_opn)
      if (rtn_code == meas_st) return (FALSE);
    if (rtn_code != ACK)  {
      printf("\r\nCommand error code = %02x: %s\r\n", rtn_code,
          diag_msg[rtn_code]);
      return (fatal_flg[rtn_code]);
    }
    if (cmd_out == DISABLE_CH)
      printf("\r\nChannel disable acknowledged\r\n");
    else
      if (data_dir == OUTPUT)
        printf("\r\nOutput setup complete\r\n");
    return (FALSE);
}
void
wait_for_valid()
{
    int vdelay;

    vdelay = MAX_CMD_RESP;
    while (~*valid_flg)  {
      access_delay();
      vdelay--;
      if (vdelay <= 0)  {
        printf("\r\nData Valid Flag Timeout\r\n");
        return;     }
    }
    *valid_flg = 0;
    if (dis_flag)print_meas();
    return;
}

/* Define a function to access the CCB status waiting for a measurement and
  using a short delay between accesses (50 usec 68030 @ 25 MHz)*/
char
wait_for_meas()
{
    unsigned int mdelay;
    unsigned char astatus;
    char meas_rtn, *diag_str;

    mdelay = MAX_MWAIT;
```

```
    if (astatus == 0xff)
      printf("\r\nWaiting for qpc angle delta completion\n");
    meas_rtn = FALSE;
    while (mdelay)                {
      mdelay--;
      if (mdelay <= 0)break;
      access_delay();
      astatus = *mstatus;
      if (astatus != 0)           {
        *mstatus = 0;
        if (astatus == meas_st)     {
          *valid_flg = 0;
          if (dis_flag) print_meas();
        }
        else {
          if (meas_st == 0xff) diag_str = timer_str;
          else if (astatus > 22)diag_str = unk_str;
             else diag_str = diag_msg[astatus];
          if (meas_cmd == MEAS_QUAD_POS_FLT)
            diag_str = quad_diag[astatus];
          printf("\r\nError status: %02x - %s", astatus,
              diag_str);
        }
        break;
      }
    }
    if (meas_st == 0xff) meas_rtn = FALSE;
    else meas_rtn = fatal_flg[astatus];
    if (mdelay <= 0)    {
      printf("\r\nNo response from measurement command\r\n");
      meas_rtn = TRUE;  }
    return(meas_rtn);
}

/* Define a function which waits for a non_zero command status with a short
  spin delay between access (~50 usec for the program development environ.) */
char
poll_resp_code(void)
{
    unsigned char p_resp;
    unsigned char *r_resp;
    int acc_att;

    r_resp = &brd_ptr->response;
    p_resp = *r_resp;
    acc_att = 0;
    while (acc_att < MAX_CMD_RESP) {
      acc_att++;
      ncounts = MIN_DELAY;
      while (--ncounts);
```

```
      if (*r_resp != p_resp)
        p_resp = *r_resp;
      else
        return (char )p_resp;  }
    exit();
}

/* Define a spin loop access delay function */
void
access_delay(void)
{
    ncounts = MIN_DELAY;
    while (--ncounts);
}

/* define a function to print any measurement base on its data type;
  unsigned short, unsigned long, long, and float. */
void
print_meas(void)
{
    printf("\r\nmeasurement = ");
    if (data_typ == INT_16) printf("%d", *(unsigned short *)meas_addr);
    else if (data_typ == INT_32) printf("%lu", *(int *)meas_addr);
    else if (data_typ == SINT_32) printf("%ld", *(unsigned int *)meas_addr);
    else if (data_typ == FLT_32) printf("%f", *(float *)meas_addr);
    else printf("%ld ang1  %ld ang2", *slvalue, *slvalue1);
    *valid_flg = 0;
}

char scr_buf[80];
int str_rtn, prt_rtn;

/* Define a function to prompt for an long integer value */
int
prompti(char *prm_str)
{
    unsigned int i_value;
    char exit_chr;

re_prmi:
    fast_print(prm_str);
    gets(scr_buf);
    exit_chr = scr_buf[0] | 0x20;

/* Test for the "x" and "r" characters to support exit and restart returns */
    if (exit_chr == 'x') return (EXIT);
    if (exit_chr == 'r') return (RESTART);
    prt_rtn = sscanf(scr_buf, "%ld", &i_value);
    if (prt_rtn != 1)
      goto re_prmi;
```

```
    return(i_value);
}

/* Define a function to prompt for a floating pt. input value */
float
promptf(char *prm_str)
{
    float f_value;

re_prmf:
    fast_print(prm_str);
    gets(scr_buf);
    prt_rtn = sscanf(scr_buf, "%f", &f_value);
    if (prt_rtn == -1)
      goto re_prmf;
    return(f_value);
}
```