VMIVME-7698

Single Board Celeron® Processor-Based VMEbus CPU

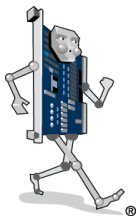
Product Manual

A GE Fanuc Company

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

500-007698-000 Rev. F

COPYRIGHT AND TRADEMARKS



(I/O man figure)

(IOWorks man figure)

Table of Contents

List of Figures

List of Tables

Overview

Introduction

VMIC's VMIVME-7698 is a complete IBM PC / AT-compatible Celeron processor-based computer with the additional benefits of single Eurocard construction and full compatibility with the VMEbus Specification Rev. C.1. The VMIVME-7698, with advanced VMEbus interface and RAM that is dual-ported to the VMEbus, is ideal for multiprocessor applications.

The single CPU board functions as a standard PC / AT, executing a PC / AT-type power-on self-test, then boots up MS-DOS, Windows 95, Windows NT, or any other PC / AT-compatible operating system. The PC / AT mode of the VMIVME-7698 is discussed in Chapter 3 of this manual.

The VMIVME-7698 also operates as a VMEbus controller and interacts with other VMEbus modules via the on-board PCI-to-VMEbus bridge and the Endian conversion hardware.

The VMIVME-7698 may be accessed as a VMEbus slave board. The VMEbus functions are available by programming the VMIVME-7698's PCI-to-VMEbus bridge according to the references defined in this volume and / or in the second volume dedicated to the optional PCI-to-VMEbus interface board titled: *VMIVME-7698 Tundra Universe II™-Based VMEbus Interface Product Manual (document No. 500-007698-001 Rev. A).*

The VMIVME-7698 programmer may quickly and easily control all the VMEbus functions simply by linking to a library of VMEbus interrupt and control functions. This library is available with VMIC's VMISFT-9420 IOWorks Access software for Windows NT users.

The VMIVME-7698 also provides capabilities beyond the features of a typical PC / AT-compatible CPU including general-purpose timers, a programmable watchdog timer, a bootable flash disk system, and nonvolatile, battery-backed SRAM. These features make the unit ideal for embedded applications. These nonstandard PC / AT functions are discussed in Chapter 4 of this manual.

Organization of the Manual

This manual is composed of the following chapters and appendices:

Chapter 1 - VMIVME-7698 Features and Options describes the features of the base unit followed by descriptions of the associated features of the unit in operation on a VMEbus.

Chapter 2 - Installation and Setup describes unpacking, inspection, hardware jumper settings, connector definitions, installation, system setup, and operation of the VMIVME-7698.

Chapter 3 - PC/AT Functions describes the unit design in terms of the standard PC memory and I/O maps, along with the standard interrupt architecture.

Chapter 4 - Embedded PC/RTOS Features describes the unit features that are beyond standard PC/AT functions.

Chapter 5 - Maintenance provides information relative to the care and maintenance of the unit.

Appendix A - Connector Pinouts illustrates and defines the connectors included in the unit's I/O ports.

Appendix B - System Drive Software includes detailed instructions for the installation of the drivers during installation of Windows 95, or Windows NT (Version 4.0) operating systems.

Appendix C - Phoenix BIOS describes the menus and options associated with the Phoenix BIOS.

Appendix D - Device Configuration: I/O and Interrupt Control provides the user with the information needed to develop custom applications such as the revision of the current BIOS configuration to a user-specific configuration.

Appendix E - Sample C Software provides a library of sample code the programmers may utilize to build the required application software for their system.

References

For the most up-to-date specifications for the VMIVME-7698, please refer to:

VMIC specification number 800-007698-000

The following books refer to the Tundra Universe II-based interface available in the VMIVME-7698:

VMIVME-7698, Tundra Universe II™-Based VMEbus Interface Product Manual
VMIC Doc. No. 500-007698-001

VMEbus Interface Components Manual
Tundra Semiconductor Corporation
603 March Rd.
Kanata, Ontario
Canada, K2K 2M5
(613) 592-0714 FAX (613) 592-1320
www.tundra.com

Some reference sources helpful in using or programming the VMIVME-7698 include:

Celeron Processors and Related Products
Intel Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
(800) 548-4752
www.intel.com

Intel 440BX PCIset
82443BX PCI and Memory Controller (PMC) 82443BX Data Bus Accelerator (DBX)
May 1996, Order Number 290549-001
Intel Corporation
P.O. Box 58119
Santa Clara, CA 95052-8119
(408) 765-8080
www.intel.com

PCI Special Interest Group
P.O. Box 14070
Portland, OR 97214
(800) 433-5177 (U.S.) (503) 797-4207 (International) FAX (503) 234-6762
www.pcisig.com

The VMEbus interrupt and control software library references included for Windows NT:

VMISFT-9420 IOWorks Access User's Guide
Doc. No. 520-009420-910
VMIC
12090 South Memorial Parkway
Huntsville, AL 35803-3308
(800) 322-3616 FAX: (256) 882-0859
www.vmic.com

For a detailed description and specification of the VMEbus, please refer to:

VMEbus Specification Rev. C.1 and The VMEbus Handbook

VMEbus International Trade Association (VITA)
7825 East Gelding Drive
Suite No. 104
Scottsdale, AZ 85260
(602) 951-8866 FAX: (602) 951-0720
www.vita.com

The following is useful information related to remote ethernet booting of the VMIVME-7698:

Microsoft Windows NT Server Resource Kit

Microsoft Corporation
ISBN: 1-57231-344-7
www.microsoft.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.

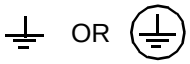
Dangerous Procedure Warnings

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

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

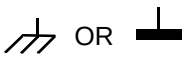
Safety Symbols Used in This Manual

Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 V are so marked).



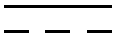
Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating equipment.

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 the 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.

Notation and Terminology

This product bridges the traditionally divergent worlds of Intel-based PC's and Motorola-based VMEbus controllers; therefore, some confusion over "conventional" notation and terminology may exist. Every effort has been made to make this manual consistent by adhering to conventions typical for the Motorola/VMEbus world; nevertheless, users in both camps should review the following notes:

- Hexadecimal numbers are listed Motorola-style, prefixed with a dollar sign: $F79, for example. By contrast, this same number would be signified 0F79H according to the Intel convention, or 0xF79 by many programmers. Less common are forms such as $F79_h$ or the mathematician's $F79_{16}$.
- An 8-bit quantity is termed a "byte," a 16-bit quantity is termed a "word," and a 32-bit quantity is termed a "longword." The Intel convention is similar, although their 32-bit quantity is more often called a "doubleword."
- Motorola programmers should note that Intel processors have an I/O bus that is completely independent from the memory bus. Every effort has been made in the manual to clarify this by referring to registers and logical entities in I/O space by prefixing I/O addresses as such. Thus, a register at "I/O $140" is not the same as a register at "$140," since the latter is on the memory bus while the former is on the I/O bus.
- Intel programmers should note that addresses are listed in this manual using a linear, "flat-memory" model rather than the old segment:offset model associated with Intel Real Mode programming. Thus, a ROM chip at a segment:offset address of C000:0 will be listed in this manual as being at address $C0000. For reference, here are some quick conversion formulas:

Segment:Offset to Linear Address

Linear Address = (Segment × 16) + Offset

Linear Address to Segment:Offset

Segment = ((Linear Address ÷ 65536) − *remainder*) × 4096

Offset = *remainder* × 65536

Where *remainder* = the fractional part of (Linear Address ÷ 65536)

Note that there are many possible segment:offset addresses for a single location. The formula above will provide a unique segment:offset address by forcing the segment to an even 64 Kbyte boundary, for example, $C000, $E000, etc. When using this formula, make sure to round the offset calculation properly.

CHAPTER 1

VMIVME-7698 Features and Options

Contents

Introduction

The VMIVME-7698 performs all the functions of a standard IBM PC / AT motherboard with the following features:

- Single-slot VMEbus 6U size
- Includes a high-performance Intel Celeron processor
- Up to 256 Mbyte of Synchronous DRAM
- 64-bit AGP SVGA video graphics accelerator
  - 2 Mbyte SGRAM Video Memory
  - Resolutions up to 1,280x1024x256 colors
- Battery-backed clock / calendar
- Front panel reset switch and miniature speaker
- On-board port for a keyboard and mouse, Ultra-IDE hard drive, floppy drive, Ethernet, video, dual serial
- Front panel “vital sign” indicators (power, Ultra-IDE hard drive activity, VMEbus SYSFAIL, and Ethernet status)
- Three general-purpose programmable 16-bit timers
- Software-controlled watchdog timer
- Up to 96 Mbyte of bootable flash on secondary IDE
- 32 Kbyte of battery-backed SRAM
- PMC, PC-MIP

The VMIVME-7698 supports standard PC / AT I / O features such as those listed in Table 1-1. Figure 1-1 on page 25 shows a block diagram of the VMIVME-7698 emphasizing the I / O features including the PCI-to-VMEbus bridge.

Table 1-1 PC/AT I/O Features

I/O FEATURE	IDENTIFIER	PHYSICAL ACCESS
Two Serial Ports (16550-Compatible RS-232C)	COM1 COM2	Front Panel, Dual Micro-D 9-Pin
AT-Style Keyboard / Mouse Controller with a PS / 2-Style Adapter	M / K	Front Panel PS / 2-Style Connector, Mini-DIN Circular (female) Adapter “Y” Cable Supplied
Super VGA Video Controller with 2 Mbyte SGRAM	SVGA	Front Panel DB15HD High Density (female)
Ethernet, 10BaseT, 100BaseTx, Novell NE-2000 Compatible	LAN	Front Panel RJ45
Floppy Disk Controller (two drives maximum)	Drives A, B	P2
Ultra IDE Fixed Disk Controller (two drives maximum)	Drives C, D	P2
Hardware Reset	RST	Front Panel Push-Button
IBM / PC Sound		Front Panel Speaker Port
Power Status, Hard Drive Activity, VMEbus SYSFAIL, and Ethernet Status	LED Indicators	Front Panel

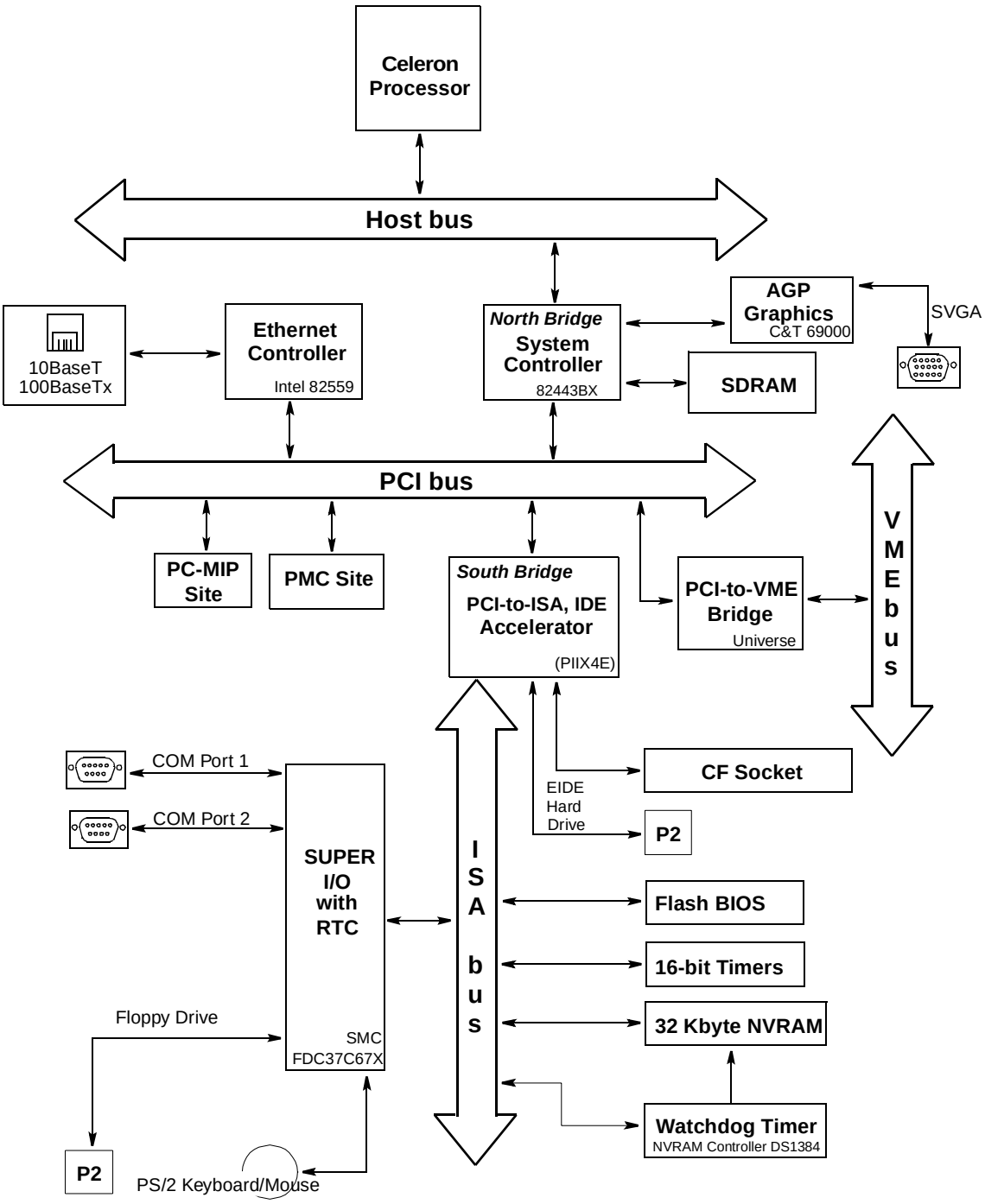


Figure 1-1 VMIVME-7698 Block Diagram

VMEbus Features

In addition to its PC / AT functions, the VMIVME-7698 has the following VMEbus features:

- Single-slot, 6U height VMEbus board
- Complete six-line Address Modifier (AM-Code) programmability
- VME data interface with separate hardware byte / word swapping for master and slave accesses
- Support for VME64 multiplexed MBLT 64-bit VMEbus block transfers
- User-configured interrupter
- User-configured interrupt handler
- System Controller mode with programmable VMEbus arbiter (PRI, SGL, and RRS modes are supported)
- VMEbus BERR* bus error timer (software programmable)
- Slave access from the VMEbus to local RAM and mailbox registers
- Full-featured programmable VMEbus requester (ROR, RWD, and BCAP modes are supported)
- System Controller autodetection
- Complete VMEbus master access through five separate Protected-mode memory windows

Figure 1-2 illustrates the VMIVME-7698 functions in a typical VMEbus system. The VMIVME-7698 is a versatile single-board solution for VMEbus control with familiar PC / AT operation.

The VMIVME-7698 VMEbus interface is provided by the PCI-to-VMEbus bridge built around the Tundra Semiconductor Corporation Universe II VMEbus interface chip. The Universe II provides a reliable high-performance 64-bit VMEbus-to-PCI interface in one design. The functions and programming of the Universe-based VMEbus interface are addressed in detail in a separate associated manual titled: The *VMIVME-7698 Tundra Universe II Based VMEbus Interface Product Manual*.

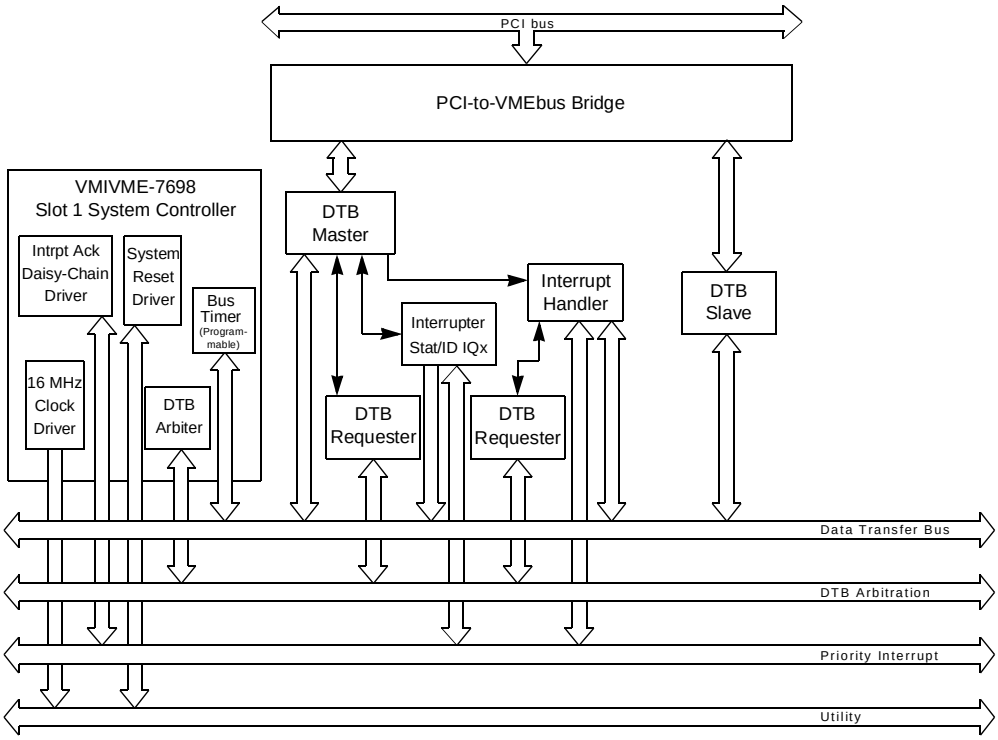


Figure 1-2 VMIVME-7698 VMEbus Functions

VMIVME-7698 Product Options

VMIC's VMIVME-7698 is built around three fundamental hardware configurations. These involve processor performance, the Flash Disk, and SDRAM memory size. *These options are subject to change based on emerging technologies and availability of vendor configurations.*

The options and current details available with the VMIVME-7698 are defined in the device specification sheet available from your VMIC representative.

CHAPTER
2

Installation and Setup

Contents

Introduction

This chapter describes the hardware jumper settings, connector definitions, installation, system setup, and operation of the VMIVME-7698. The PCI-to-VMEbus bridge and the Tundra Universe II-based interface are also included.

Unpacking Procedures

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. All claims arising from shipping damage should be filed with the carrier and a complete report sent to VMIC Customer Service together with a request for advice concerning the disposition of the damaged item(s).

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 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.

Hardware Setup

The VMIVME-7698 is factory populated with user-specified options as part of the VMIVME-7698 ordering information. The CPU speed, SDRAM, and flash size are not user-upgradable. To change these options contact customer service to receive a Return Material Authorization (RMA).

VMIC Customer Service is available at: 1-800-240-7782.
Or E-mail us at customer.service@vmic.com

The VMIVME-7698 is tested for system operation and shipped with factory-installed header jumpers. The physical location of the jumpers and connectors for the single board CPU are illustrated in Figure 2-1 on page 31. The definitions of the CPU board jumpers and connectors are included in Table 2-1 through Table 2-4. The Tundra Universe II-based PCI-to-VMEbus Bridge jumper configuration is discussed in Chapter 3.

CAUTION: All jumpers are factory configured and should not be modified by the user. There are three exceptions: the Password Clear (E4), the Programmable Timer Clock Select (E8), the Watchdog Timer (E8) and the NVRAM Battery Power (E8).

Modifying any other jumper will void the Warranty and may damage the unit. The default jumper condition of the VMIVME-7698 is expressed in Table 2-1 through Table 2-7 with **bold text** in the table cells.

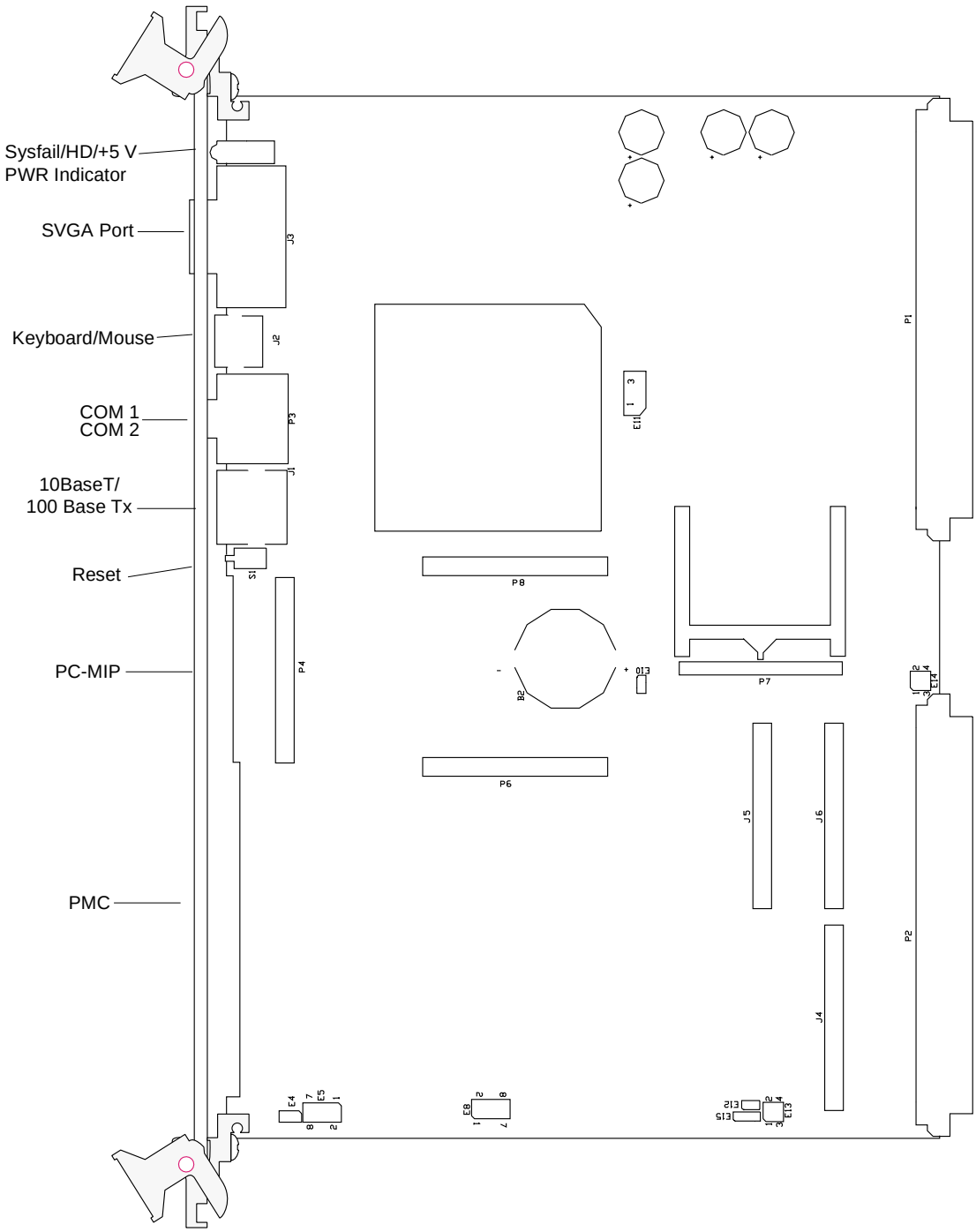


Figure 2-1 VMIVME-7698 CPU Board, I/O Port, and Jumper Locations

Table 2-1 CPU Board Connectors

Connector	Function
P3	COM1 / COM2
J3	Video Connector
J1	Ethernet Connector
J2	PS / 2 Keyboard / Mouse Connector
E11	Fan Connector
P1, P2	VME Connectors
J4, J5, J6	PMC PCI Expansion Connectors
P4, P6, P8	PC-MIP PCI Expansion Connectors

Jumper E8 below serves several functions on the VMIVME-7698. These functions are: The Watchdog Timer / NMI, Programmable Timer Clock Select, and the NVRAM Battery Enable.

Table 2-2 Timers and NVRAM Battery Select (**User Configurable**) - Jumper (E8)

Select	Jumper Position
Watchdog Timer	
RESET	**2-4**
NMI	4-6
No RESET or NMI	Out
Programmable Timer Clock Selection	
1 Mhz	**1-3**
2 Mhz	3-5
***NVRAM Battery Power**	
NVRAM Battery Disabled	Out
NVRAM Battery Enabled	**7-8**

**Enable battery for operation, disable battery for storage*

Table 2-3 BIOS Mode Option - Jumper (E15)

Select	Jumper Position
Normal	**1-2**
Boot Block Programming	**2-3**

NOTE: The VMIVME-7698's BIOS has the capability (Default: Disabled) of password protecting casual access to the unit's CMOS set-up screens. The Password Clear jumper (E4) allows for a means to clear the password feature, as might be necessary to do in the case of a forgotten password.

To clear the CMOS:

1. Turn off power to the unit.
2. Install a jumper at E4.
3. Power up the unit.
4. Turn off the power to the unit and remove the jumper from E4.

When power is re-applied to the unit, the CMOS will be cleared.

Table 2-4 Password Clear (**User Configurable**) - Jumper (E4)

	Jumper Position
Normal	**Out**
Clear CMOS / Password	In

Table 2-5 CPU Clock/Bus Multiple - Jumper Blocks (E5)

CPU Option	Jumper Position
Celeron 300 MHz	3-4, 7-8
Celeron 366 MHz	OUT

Table 2-6 CMOS Battery Enable - Jumper (E10)

	Jumper Position
CMOS Battery Disabled	Out
CMOS Battery Enabled	**In**

Table 2-7 VME Interface Jumper Options and Factory Settings

Jumper	Function	Jumper Position
E13, 1-3	Installed - Universe Memory Mapped	Installed (should not be removed)
E13, 2-4	Installed - SYSFAIL Not Asserted Upon Reset Removed - SYSFAIL Asserted Upon Reset	Installed
E14, 1-3	Installed - Drives VMEbus SYSRESET Removed - Does not drive	Installed
E14, 2-4	Installed - Receives VMEbus SYSRESET Removed - Does not receive	Installed

Installation

The VMIVME-7698 conforms to the VMEbus physical specification for a 6U dual Eurocard (dual height). It can be plugged directly into any standard chassis accepting this type of board.

CAUTION: Do not install or remove the board while power is applied.

The following steps describe the VMIC-recommended method for VMIVME-7698 installation and power-up:

1. Make sure power to the equipment is off.
2. Choose chassis slot. The VMIVME-7698 ***must*** be attached to a dual P1/P2 VMEbus backplane.

   If the VMIVME-7698 is to be the VMEbus system controller, choose the first VMEbus slot. If some other board is the VMEbus system controller, choose any slot **except** slot one. The VMIVME-7698 does not require jumpers for enabling/disabling the system controller function.

NOTE: The VMIVME-7698 requires forced air cooling. It is advisable to install blank panels over any exposed VMEbus slots. This will allow for better air flow over the VMIVME-7698 board. For 20-slot VME configurations, three 100 CFM fans are recommended.

3. Insert the VMIVME-7698 and its attached expansion modules into the chosen VMEbus chassis slot (expansion modules should fill the slots immediately adjacent to the VMIVME-7698). While ensuring that the boards are properly aligned and oriented in the supporting board guides, slide the boards smoothly forward against the mating connector until firmly seated.
4. Connect all needed peripherals to the front panel. Each connector is clearly labeled on the front panel, and detailed pinouts are in Appendix A. Minimally, a keyboard and a monitor are required if the user has not previously configured the system.
5. Apply power to the system. Several messages are displayed on the screen, including names, versions, and copyright dates for the various BIOS modules on the VMIVME-7698.
6. The VMIVME-7698 features a Flash Disk resident on the board. Refer to Chapter 4 for set up details.
7. If an external drive module is installed, the BIOS Setup program must be run to configure the drive types. See Appendix C to properly configure the system.

8. If a drive module is present, install the operating system according to the manufacturer's instructions.

See Appendix B for instructions on installing VMIVME-7698 peripheral driver software during operating system installation.

BIOS Setup

The VMIVME-7698 has an on-board BIOS Setup program that controls many configuration options. These options are saved in a special nonvolatile, battery-backed memory chip and are collectively referred to as the board's "CMOS configuration." The CMOS configuration controls many details concerning the behavior of the hardware from the moment power is applied.

The VMIVME-7698 is shipped from the factory with no hard drives configured in CMOS. The BIOS Setup program must be run to configure the specific drives attached.

Details of the VMIVME-7698 BIOS setup program are included in Appendix C.

Front Panel Connectors

The front panel connections, including connector pinouts and orientation, for the VMIVME-7698 are defined in detail in Appendix A.

PMC Expansion Site Connectors

The VMIVME-7698 supplies PMC and PC-MIP expansion site connectors for adding a PMC and/or PC-MIP expansion board. This expansion capability allows third-party devices to be used with the VMIVME-7698, as shown in Figure 2-2.

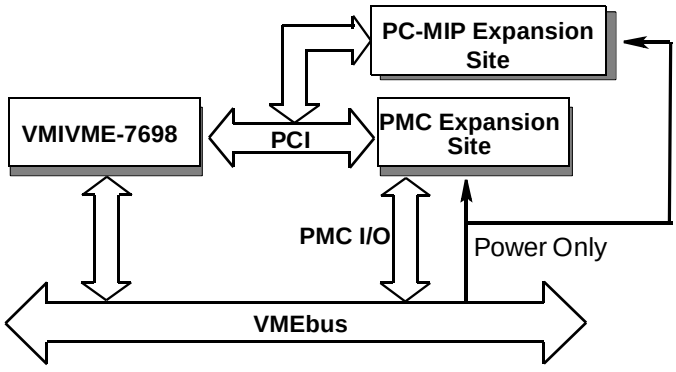
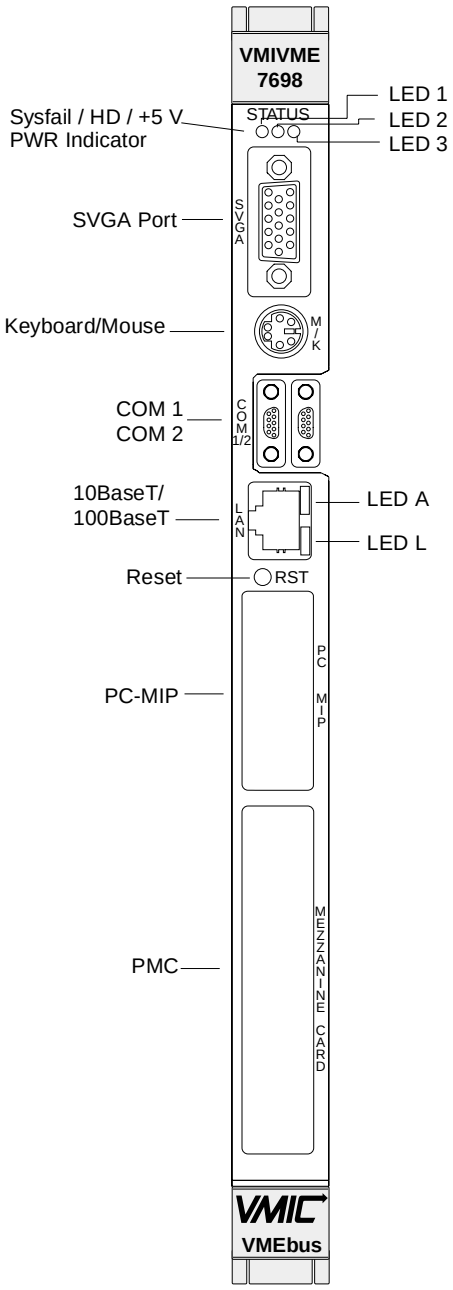


Figure 2-2 PCI Expansion Site

LED Definition



LED 1 *Power* - Indicates when power is applied to the board.

LED 2 *Hard Drive Indicator* - Indicates when hard drive activity is occurring.

LED 3 *SYSFAIL* - Indicates when a VMEbus SYSFAIL is asserted.

LED A *Ethernet Active* - Indicates when the Ethernet is transmitting or receiving data.

LED L *Ethernet Link* - *Yellow* indicates when the Ethernet is linked in 10BaseT mode. *Green* indicates when the Ethernet is linked in 100BaseTx mode.

Figure 2-3 LED Position on the Front Panel

CHAPTER
3

PC/AT Functions

Contents

Introduction

The VMIVME-7698 provides a complete IBM PC / AT-compatible Celeron processor-based computer. The design includes a high-speed microprocessor with current technology memory. Reference the VMIC product specifications for available component options.

Because the design is PC / AT compatible, it retains standard PC memory and I / O maps along with standard interrupt architecture. Furthermore, the VMIVME-7698 includes a PCI-compatible video adapter and Ethernet controller.

The following sections describe in detail the PC / AT functions of the VMIVME-7698.

CPU Socket

The VMIVME-7698 CPU socket is factory populated with a high-speed Celeron processor. The CPU speed and SDRAM size are user-specified as part of the VMIVME-7698 ordering information. The options are not user-upgradable.

To change CPU speeds or RAM size, contact VMIC customer service to obtain a Return Material Authorization (RMA).

VMIC Customer Service is available at: 1-800-240-7782.

Or E-mail us at customer.service@vmic.com

Physical Memory

The VMIVME-7698 provides Synchronous DRAM (SDRAM) as on-board system memory. Memory can be accessed as bytes, words, or longwords.

All RAM on the VMIVME-7698 is dual-ported to the VMEbus through the PCI-to-VME bridge. The memory is addressable by the local processor, as well as the VMEbus slave interface by another VMEbus master. Caution must be used when sharing memory between the local processor and the VMEbus to prevent a VMEbus master from overwriting the local processor's operating system.

NOTE: When using the Configure utility of I/O Works Access with Windows NT 4.0 to configure RAM, do not request more than 25 percent of the physical RAM. Exceeding the 25 percent limit may result in a known Windows NT bug causing unpredictable behavior during the Windows NT boot sequence and require the use of an emergency repair disk to restore the computer. The bug is present in Windows NT 4.0 service pack level 3. It is recommended that an emergency repair disk be kept up-to-date and easily accessible.

The VMIVME-7698 includes the system BIOS, video BIOS, and LANWorks BIOS in a single flash memory device.

The VMIVME-7698 memory includes 32 Kbyte of battery-backed SRAM addressed at $D8000 to $DFFFF. All but the first 24 bytes are accessible. Bytes $D8000 - $D800D are used by the Watchdog controller and include the System Command Register (COMM), the VME BERR Address Register (VBAR), the VME BERR Address Modifier Register (VBAMR), and the Board ID Register (BID). The System registers are explained in Volume II of this manual set. The battery-backed SRAM can be accessed by the CPU at anytime, and can be used to store system data that must not be lost during power-off conditions (Bytes $D800E - $D8017 are the System Registers).

Memory and Port Maps

Memory Map

The memory map for the VMIVME-7698 is shown in Table 3-1. All systems share this same memory map, although a VMIVME-7698 with less than the full 256 Mbyte of SDRAM does not fill the entire space reserved for On-Board Extended Memory.

Table 3-1 VMIVME-7698, Universe II-Based Interface Memory Address Map

MODE	MEMORY ADDRESS RANGE	SIZE	DESCRIPTION
PROTECTED MODE	$FFFF 0000 - $FFFF FFFF	64 Kbyte	ROM BIOS Image
	$0400 0000 - $FFFE FFFF	3.9 Gbyte	Unused *
	$0010 0000 - $0FFF FFFF	255 Mbyte	Reserved for ** On-Board Extended Memory (not filled on all systems)
REAL MODE	$E0000 - $FFFFF	128 Kbyte	ROM BIOS
	$D8018 - $DFFFF	32 Kbyte minus 24 bytes	Battery-Backed SRAM
	$D8016 - $D8017	2 bytes	Board ID Register (BID) (0x7698)
	$D8014 - $D8015	2 bytes	VME BERR Address Modifier Register (VBAMR)
	$D8010 - $D8013	4 bytes	VME BERR Address Register (VBAR)
	$D800E - $D800F	2 bytes	System Command Register (COMM)
	$D8000 - $D800D	14 bytes	RTC / Watchdog Timer Control Registers
	$C8000 - $D7FFF	64 Kbyte	LANWorks BIOS
	$C0000 - $C7FFF	32 Kbyte	Video ROM
	$A0000 - $BFFFF	128 Kbyte	Video RAM
	$00000 - $9FFFF	640 Kbyte	User RAM / DOS RAM

* This space can be used to set up protected mode PCI-to-VMEbus windows (also referred to as PCI slave images).

** This space can be allocated as shared memory (for example, between the Celeron processor-based CPU and VMEbus Master. Note, that if a PMC board is loaded, the expansion BIOS may be placed in this area.

I/O Port Map

The Celeron processor-based CPU includes special input/output instructions that access I/O peripherals residing in I/O addressing space (separate and distinct from memory addressing space). Locations in I/O address space are referred to as *ports*. When the CPU decodes and executes an I/O instruction, it produces a 16-bit I/O address on lines A00 to A15 and identifies the I/O cycle to the processor's M/IO control line. Thus, the CPU includes an independent 64Kbyte I/O address space, which is accessible as bytes, words, or longwords.

Standard PC/AT hardware circuitry reserves only 1,024 bytes of I/O addressing space from I/O $000 to $3FF for peripherals. All standard PC I/O peripherals such as serial and parallel ports, hard and floppy drive controllers, video system, real-time clock, system timers, and interrupt controllers are addressed in this region of I/O space. The BIOS initializes and configures all these registers properly; adjusting these I/O ports directly is not normally necessary.

The assigned and user-available I/O addresses are summarized in the I/O Address Map, Table 3-2.

Table 3-2 VMIVME-7698 I/O Address Map

I/O ADDRESS RANGE	SIZE IN BYTES	HW DEVICE	PC/AT FUNCTION
$000 - $00F	16		DMA Controller 1 (Intel 8237A Compatible)
$010 - $01F	16		Reserved
$020 - $021	2		Master Interrupt Controller (Intel 8259A Compatible)
$022 - $03F	30		Reserved
$040 - $043	4		Programmable Timer (Intel 8254 Compatible)
$044 - $05F	30		Reserved
$060 - $064	5		Keyboard, Speaker, Eqpt. Configuration (Intel 8042 Compatible)
$065 - $06F	11		Reserved
$070 - $071	2		Real-Time Clock, NMI Mask
$072 - $07F	14		Reserved
$080 - $08F	16		DMA Page Registers
$090 - $091	2		Reserved
$092	1		Alt. Gate A20/Fast Reset Register
$093	1		Reserved

Table 3-2 VMIVME-7698 I/O Address Map (Continued)

I/O ADDRESS RANGE	SIZE IN BYTES	HW DEVICE	PC/AT FUNCTION
$094	1	Super VGA Chip	POS102 Access Control Register
$095 - $09F	11		Reserved
$0A0 - $0A1	2		Slave Interrupt Controller (Intel 8259A Compatible)
$0A2 - $0BF	30		Reserved
$0C0 - $0DF	32		DMA Controller 2 (Intel 8237A Compatible)
$0E0 - $16F	142		Reserved
$170 - $177	8	PIIX4E	Secondary Hard Disk Controller
$178 - $1EF	120		User I/O
$1F0 - $1F7	8	PIIX4E	Primary Hard Disk Controller
$1F8 - $277	128		User I/O
$278 - $27F	8	Super I/O Chip*	LPT2 Parallel I/O*
$280 - $2E7	104		Reserved
$2E8 - $2EE	7	UART*	COM4 Serial I/O*
$2EF - $2F7	9		User I/O
$2F8 - $2FE	7	Super-I/O Chip	COM2 Serial I/O (16550 Compatible)
$2FF - $36F	113		Reserved
$370 - $377	8	Super-I/O Chip	Secondary Floppy Disk Controller
$378 - $37F	8	Super-I/O Chip	LPT1 Parallel I/O
$380 - $3E7	108		Reserved
$3E8 - $3EE	7	UART*	COM3 Serial I/O*
$3F0 - $3F7	8	Super-I/O Chip	Primary Floppy Disk Controller
$3F8 - $3FE	7	Super-I/O Chip	COM1 Serial I/O (16550 Compatible)
$3FF - $4FF			Reserved
$500 - $503	4	82C54 Timer	Programmable Internal Timer
$504 - $CFF			Reserved

* While these I/O ports are reserved for the listed functions, they are not implemented on the VMIVME-7698. They are listed here to make the user aware of the standard PC/AT usage of these ports.

PC/AT Interrupts

In addition to an I/O port address, an I/O device has a separate hardware interrupt line assignment. Assigned to each interrupt line is a corresponding interrupt vector in the 256-vector interrupt table at $00000 to $003FF in memory. The 16 maskable interrupts and the single Non-Maskable Interrupt (NMI) are listed in Table 3-3 along with their functions. Table 3-4 on page 45 details the vectors in the interrupt vector table. The interrupt number in HEX and decimal are also defined for real and protected mode in Table 3-4.

The interrupt hardware implementation on the VMIVME-7698 is standard for computers built around the PC/AT architecture, which evolved from the IBM PC/XT. In the IBM PC/XT computers, only eight interrupt request lines exist, numbered from IRQ0 to IRQ7 at the PIC. The IBM PC/AT computer added eight more IRQx lines, numbered IRQ8 to IRQ15, by cascading a second slave PIC into the original master PIC. IRQ2 at the master PIC was committed as the cascade input from the slave PIC. This architecture is represented in Figure 3-1 on page 49.

To maintain backward compatibility with PC/XT systems, IBM chose to use the new IRQ9 input on the slave PIC to operate as the old IRQ2 interrupt line on the PC/XT Expansion Bus. Thus, in AT systems, the IRQ9 interrupt line connects to the old IRQ2 pin (pin B4) on the AT Expansion Bus (or ISA bus).

Table 3-3 PC/AT Hardware Interrupt Line Assignments

IRQ	AT FUNCTION	COMMENTS
NMI	Parity Errors (Must be enabled in BIOS Setup)	Used by VMIVME-7698 VMEbus Interface
0	System Timer	Set by BIOS Setup
1	Keyboard	Set by BIOS Setup
2	Duplexed to IRQ9	
3	COM2/COM4	
4	COM1/COM3	
5	Timer	Assigned to On-Board Timer
6	Floppy Controller	
7	LPT1	Not Available
8	Real-Time Clock	
9	Old IRQ2	Determined by BIOS

Table 3-3 PC/AT Hardware Interrupt Line Assignments (Continued)

IRQ	AT FUNCTION	COMMENTS
10	Not Assigned	Determined by BIOS
11	Not Assigned	Determined by BIOS
12	Mouse	
13	Math Coprocessor	
14	AT Hard Drive	
15	Flash Drive	

Table 3-4 PC/AT Interrupt Vector Table

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
00	0		Divide Error	Same as Real Mode
01	1		Debug Single Step	Same as Real Mode
02	2	NMI	Memory Parity Error, VMEbus Interrupts	Same as Real Mode (Must be enabled in BIOS Setup)
03	3		Debug Breakpoint	Same as Real Mode
04	4		ALU Overflow	Same as Real Mode
05	5		Print Screen	Array Bounds Check
06	6			Invalid OpCode
07	7			Device Not Available
08	8	IRQ0	Timer Tick	Double Exception Detected
09	9	IRQ1	Keyboard Input	Coprocessor Segment Overrun
0A	10	IRQ2	BIOS Reserved	Invalid Task State Segment
0B	11	IRQ3	COM2 Serial I / O	Segment Not Present
0C	12	IRQ4	COM1 Serial I / O	Stack Segment Overrun
0D	13	IRQ5	Onboard 16bit Timers	Same as Real Mode
0E	14	IRQ6	Floppy Disk Controller	Page Fault
0F	15	IRQ7	LPT1 Parallel I / O	Unassigned

Table 3-4 PC/AT Interrupt Vector Table (Continued)

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
10	16		BIOS Video I/O	Coprocessor Error
11	17		Eqpt Configuration Check	Same as Real Mode
12	18		Memory Size Check	Same as Real Mode
13	19		XT Floppy/Hard Drive	Same as Real Mode
14	20		BIOS Comm I/O	Same as Real Mode
15	21		BIOS Cassette Tape I/O	Same as Real Mode
16	22		BIOS Keyboard I/O	Same as Real Mode
17	23		BIOS Printer I/O	Same as Real Mode
18	24		ROM BASIC Entry Point	Same as Real Mode
19	25		Bootstrap Loader	Same as Real Mode
1A	26	IRQ8	Real-Time Clock	Same as Real Mode
1B	27		Control/Break Handler	Same as Real Mode
1C	28		Timer Control	Same as Real Mode
1D	29		Video Parameter Table Pntr	Same as Real Mode
1E	30		Floppy Parm Table Pntr	Same as Real Mode
1F	31		Video Graphics Table Pntr	Same as Real Mode
20	32		DOS Terminate Program	Same as Real Mode
21	33		DOS Function Entry Point	Same as Real Mode
22	34		DOS Terminate Handler	Same as Real Mode
23	35		DOS Control/Break Handler	Same as Real Mode
24	36		DOS Critical Error Handler	Same as Real Mode
25	37		DOS Absolute Disk Read	Same as Real Mode
26	38		DOS Absolute Disk Write	Same as Real Mode
27	39		DOS Program Terminate, Stay Resident	Same as Real Mode
28	40		DOS Keyboard Idle Loop	Same as Real Mode
29	41		DOS CON Dev. Raw Output	Same as Real Mode

Table 3-4 PC/AT Interrupt Vector Table (Continued)

INTERRUPT NO.		IRQ LINE	REAL MODE	PROTECTED MODE
HEX	DEC			
2A	42		DOS 3.x+ Network Comm	Same as Real Mode
2B	43		DOS Internal Use	Same as Real Mode
2C	44		DOS Internal Use	Same as Real Mode
2D	45		DOS Internal Use	Same as Real Mode
2E	46		DOS Internal Use	Same as Real Mode
2F	47		DOS Print Spooler Driver	Same as Real Mode
30-60	48-96		Reserved by DOS	Same as Real Mode
61-66	97-102		User Available	Same as Real Mode
67-70	103-112		Reserved by DOS	Same as Real Mode
71	113	IRQ9	Not Assigned	
72	114	IRQ10	Not Assigned	
73	115	IRQ11	Not Assigned	
74	116	IRQ12	Mouse	
75	117	IRQ13	Math Coprocessor	
76	118	IRQ14	AT Hard Drive	
77	119	IRQ15	Flash Drive	
78-7F	120-127		Reserved by DOS	Same as Real Mode
80-F0	128-240		Reserved for BASIC	Same as Real Mode
F1-FF	241-255		Reserved by DOS	Same as Real Mode

PCI Interrupts

Interrupts on Peripheral Component Interconnect (PCI) Local Bus are optional and defined as “level sensitive,” asserted low (negative true), using open drain output drivers. The assertion and de-assertion of an interrupt line, INTx#, is asynchronous to CLK. A device asserts its INTx# line when requesting attention from its device driver. Once the INTx# signal is asserted, it remains asserted until the device driver clears the pending request. When the request is cleared, the device de-asserts its INTx# signal.

PCI defines one interrupt line for a single function device and up to four interrupt lines for a multifunction device or connector. For a single function device, only INTA# may be used while the other three interrupt lines have no meaning. Figure 3-1 on page 49 depicts the VMIVME-7698 interrupt logic pertaining to VMEbus operations and the PCI expansion site.

Any function on a multifunction device can be connected to any of the INTx# lines. The Interrupt Pin register defines which INTx# line the function uses to request an interrupt. If a device implements a single INTx# line, it is called INTA#; if it implements two lines, they are called INTA# and INTB#; and so forth. For a multifunction device, all functions may use the same INTx# line or each may have its own (up to a maximum of four functions) or any combination thereof. A single function can not generate an interrupt request on more than one INTx# line.

The slave PCI accepts the VMEbus interrupts through lines that are defined by the BIOS. The BIOS defines which interrupt line to utilize depending on which system requires the use of the line.

The PCI-to-VME Bridge has the capability of generating a Non-Maskable Interrupt (NMI) via the PCI SERR# line. Table 3-5 describes the register bits that are used by the NMI. The SERR interrupt is routed through certain logic back to the NMI input line on the CPU. The CPU reads the NMI Status Control register to determine the NMI source (bits set to 1). After the NMI interrupt routine processes the interrupt, software clears the NMI status bits by setting the corresponding enable / disable bit to 1. The NMI Enable and Real-Time Clock register can mask the NMI signal and disable / enable all NMI sources.

Table 3-5 NMI Register Bit Descriptions

Status Control Register (I/O Address $061, Read/Write, Read Only)	
Bit 7	SERR# NMI Source Status (Read Only) - This bit is set to 1 if a system board agent detects a system board error. It then asserts the PCI SERR# line. To reset the interrupt, set bit 2 to 0 and then set it to 1. When writing to port $061, bit 7 must be 0.
Bit 2	PCI SERR# Enable (Read / Write) - 1 = Clear and Disable, 0 = Enable
Enable and Real-Time Clock Address Register (I/O Address $070, Write Only)	
Bit 7	NMI Enable - 1 = Disable, 0 = Enable

I/O Ports

The VMIVME-7698 incorporates the SMC Super-I/O chip. The SMC chip provides the VMIVME-7698 with a standard floppy drive controller and two 16550 UART-compatible serial ports. These ports are RS-232C compliant and support the imposed standard cable length of 50 feet, maximum, in order to meet the requirements of EN61000-4-5 for CE mark. The Ultra-IDE hard drive interface is provided by the Intel 82371EB (PIIX4E) PCI ISA IDE Xcelerator chip. All ports are present in their standard PC/AT locations using default interrupts.

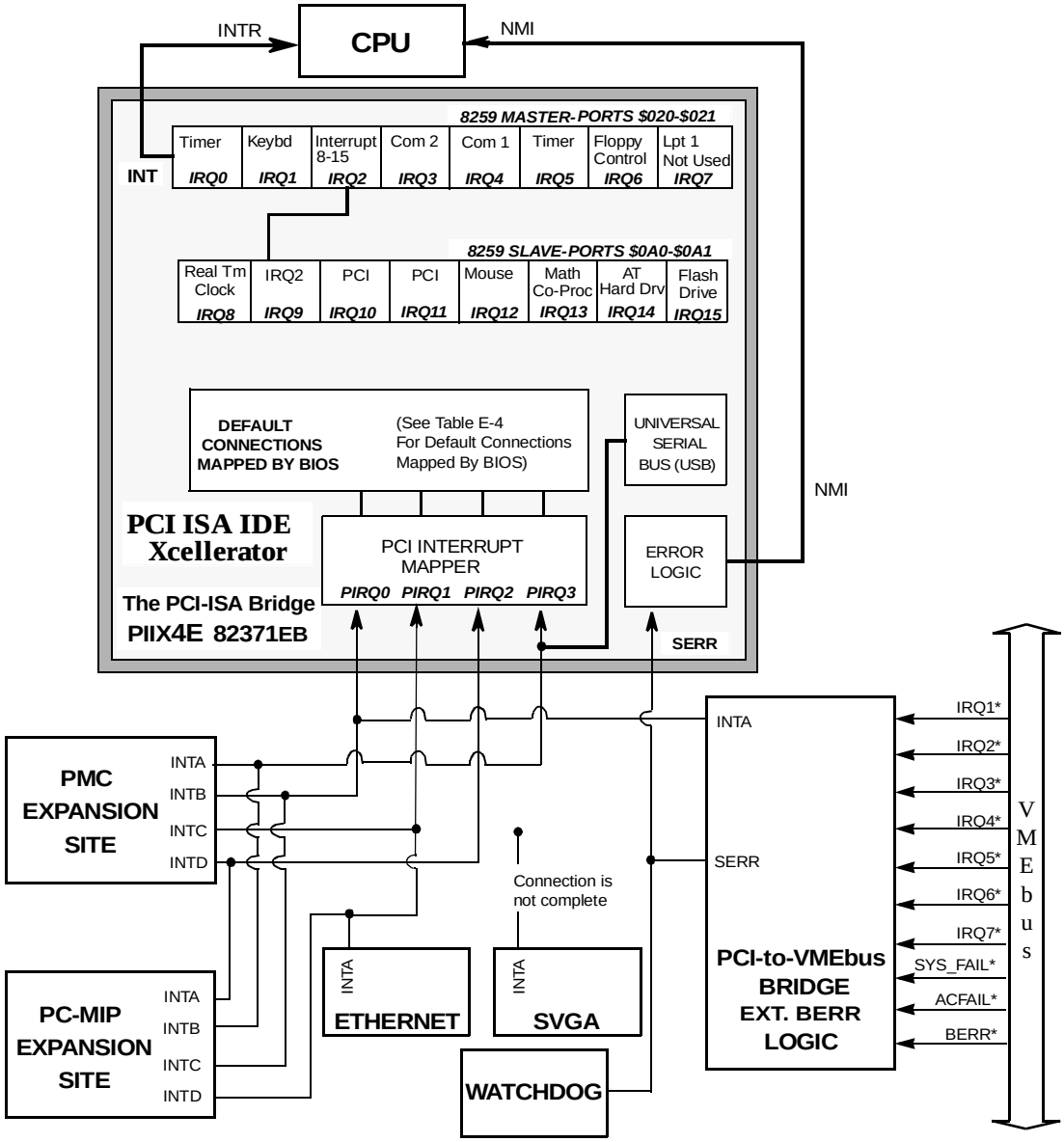


Figure 3-1 Connections for the PC Interrupt Logic Controller

Video Graphics Adapter

The monitor port on the VMIPMC-7698 is controlled by a Chips and Technology 69000 video adapter chip with 2 Mbyte video SDRAM. The video controller chip is hardware and BIOS compatible with the IBM EGA and SXGA standards and also supports VESA high-resolution and extended video modes. Table 3-6 shows the graphics video modes supported by the VMIPMC-7698.

Table 3-6 Supported Graphics Video Resolutions

SCREEN RESOLUTION	MAXIMUM COLORS	REFRESH RATES (Hz)
640 x 480	16 M	60, 75, 85
800 x 600	16 M	60, 75, 85
1024 x 768	64 K	60, 75, 85
1280 x 1024	256	60, 75

Not all SVGA monitors support resolutions and refresh rates beyond 640 x 480 at 60 Hz. Do not attempt to drive a monitor to a resolution or refresh rate beyond its capability.

Floppy disks supplied with the VMIVME-7698 also contain video drivers for Windows 3.11 and Windows operating systems. Appendix B contains instructions on the incorporation of the video drivers during operating system installation.

Ethernet Controller

The network capability is provided by Intel's 82559. This Ethernet controller is PCI bus based and is software configurable. The VMIVME-7698 supports 10BaseT and 100BaseTx Ethernet. This ethernet supports LANWorks remote ethernet boot option. Refer to Appendix C for Setup.

10BaseT

A network based on the 10BaseT standard uses unshielded twisted-pair cables, providing an economical solution to networking by allowing the use of existing telephone wiring and connectors. The RJ-45 connector is used with the 10BaseT standard. 10BaseT has a maximum length of 100 m from the wiring hub to the terminal node.

100BaseTx

The VMIVME-7698 also supports the 100BaseTx Ethernet. A network based on a 100BaseTx standard uses unshielded twisted-pair cables and a RJ-45 connector. The 100BaseTx has a maximum deployment length of 250 m.

LANWorks

The VMIVME-7698 ethernet controller supports remote booting using the LANWorks Ethernet BIOS. Refer to Appendix C for more information on remote Ethernet booting.

CHAPTER

4

Embedded PC/RTOS Features

Contents

Introduction

VMIC's VMIVME-7698 features additional capabilities beyond those of a typical IBM PC / AT-compatible CPU. The unit provides three software-controlled, general-purpose timers along with a programmable Watchdog timer for synchronizing and controlling multiple events in embedded applications. The VMIVME-7698 also provides a bootable flash disk system and 32 K byte of nonvolatile, battery-backed SRAM. These features make the unit ideal for embedded applications, particularly applications where standard hard drives and floppy disk drives cannot be used.

Timers

General

The VMIVME-7698 provides a user-programmable 82C54 internal timer/counter. The 82C54 provides three independent, 16-bit timers, each operating at 1 or 2 MHz clock speed determined by the configuration of jumper E8; reference Table 2-2 on page 32. These timers are completely available to the user and are not dedicated to any PC/AT function. These timers may be used to generate system interrupts.

Events can be timed by either polling the timers or generating a system interrupt via circuity external to the 82C54. The external circuity consists of logic which generates the interrupt and a Timer Interrupt Status register which indicates which of the three Timers generated an interrupt.

The 82C54 timers are mapped at I/O address $500. The interrupt used by the Timers is IRQ5. The Timer Interrupt Status register is available via the Power Management I/O address space. The access to this space is explained in the "Timer Interrupt Status" section below.

Timer Interrupt Status

A single interrupt, IRQ5, is used by all three Timers. A Timer Interrupt Status register is provided in order to determine which Timer(s) initiated an interrupt. The interrupt status register is a general-purpose input register located, external to the 82C54, at offset 31h from the Power Management Base I/O address. The interrupt status register address can be found by first determining the PCI Configuration Base address for Device ID 7113h and Vendor ID 8086h. The Power Management Base I/O address can be found by reading offset 40h from this PCI Configuration address. The Timer Interrupt Status register bits are located at offset 31h from the Power Management Base I/O address, bits 5, 6, and 7 (refer to Figure 4-2).

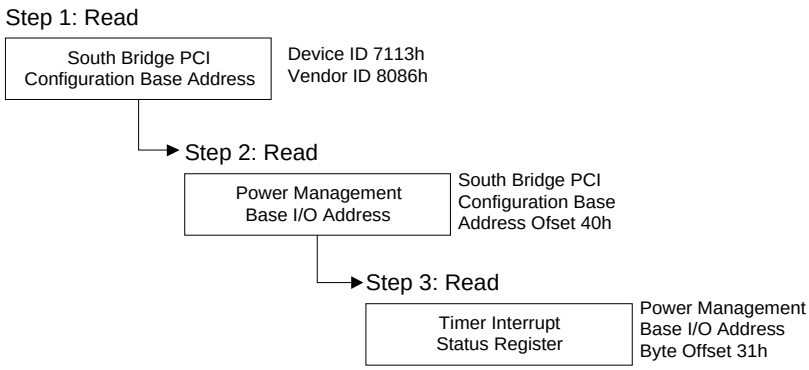


Figure 4-1 Timer Interrupt Status Register Read/Steps

A byte read of Offset 31h from the Power Management Base I/O address is used to obtain these bits. Bits 5, 6, and 7 correspond to Timers 2, 1, and 0, respectively

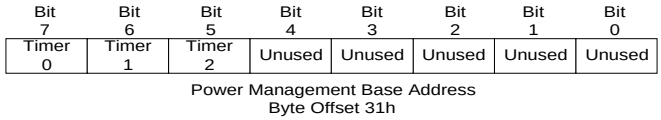


Figure 4-2 Timer Interrupt Status Register

In order to clear the Timer Interrupt Status register, first write zeros (0's) to the general-purpose output register located at offset 37h of the Power Management Base I/O address bits 3, 4, and 6 (Not bits 3, 4 and 5). Then write ones (1's) to these same bits to re-enable the Timer Interrupt Status register. Bits 3, 4, and 6 correspond to Timers 2, 1, and 0, respectively.

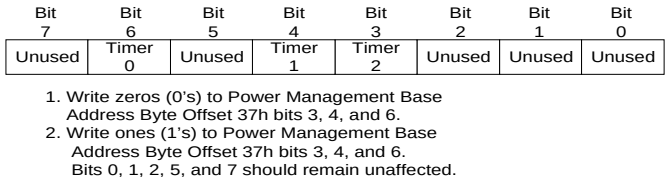


Figure 4-3 Clearing the Timer Interrupt Status Register

Clearing the Interrupt

The Timer Interrupts are cleared using the standard procedure for clearing PC/AT IRQ5. Refer to Appendix D for an example of using the 82C54 timers.

Timer Programming

Architecture

The VMIVME-7698 Timers are mapped in I/O address space starting at $500. See Table 4-1. The Timers, consisting of three 16-bit timers and a Control Word Register (see Figure 4-4) are read from/written to via an 8-bit data bus.

Table 4-1 I/O Address of the Control Word Register and Timers

I/O Address	Select
$500	Timer 0
$501	Timer 1
$502	Timer 2
$503	Control Word Register

Table 4-1 shows the I/O address's of the Control Word Register and Timers.

The Control Word Register is write only. The Timer status information can be obtained from the Read-Back command (see the "Reading" section on page 59).

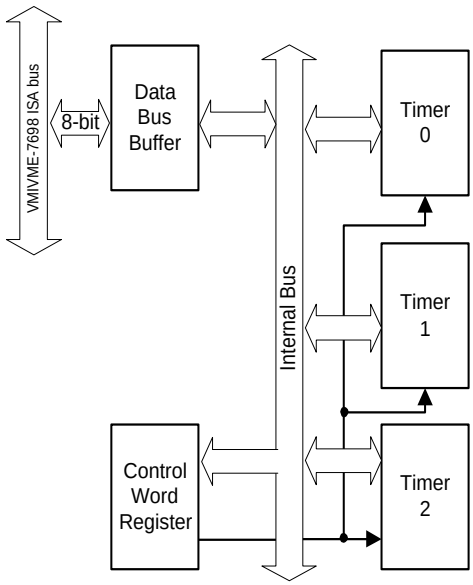


Figure 4-4 82C54 Diagram

The three Timers, Timer 0, 1, and 2, are functionally equivalent. Therefore only a single Timer will be described. Figure 4-5 is a block diagram of a Timer. Each Timer is functionally independent. Although the Control word is shown in the Timer block diagram, it is not a part of the Timer, but its contents directly affect how the Timer functions.

The status register, as shown in Figure 4-5, when latched, contains the present contents of the Control Word Register and the present state of the output and load count flag (The Status Word is available via the Read-Back command, see the "Reading" section on page 59).

The Timer is labeled TE (Timer Element). It is a 16-bit synchronous presettable down counter.

The blocks labeled OL_M and OL_L are 8-bit Output Latches (OL). The subscripts M and L stand for Most Significant byte and Least Significant byte. These latches usually track the TE but when commanded will latch and hold the present count until the CPU reads the count. When the latched count is read, the OL registers will continue to track the TE. When reading the OL registers, two 8-bit accesses must be performed to retrieve the complete 16-bit value of the Timer as only one latch at a time is enabled. The TE cannot be read; the count is read from the OL registers.

There are two 8-bit registers labeled TR_M and TR_L (Timer Register). The subscripts M and L stand for Most Significant byte and Least Significant byte. When a new count is written to the Timer, the count is loaded into the TR and later transferred to the TE. The Control logic lets one 8-bit TR register be written to at a time. Two 8-bit writes must be performed to load a complete 16-bit count value. Both TR bytes are transferred to the TE at the same time. The TE cannot be directly written to by the user, the count is written to the TR registers, then latched to the TE.

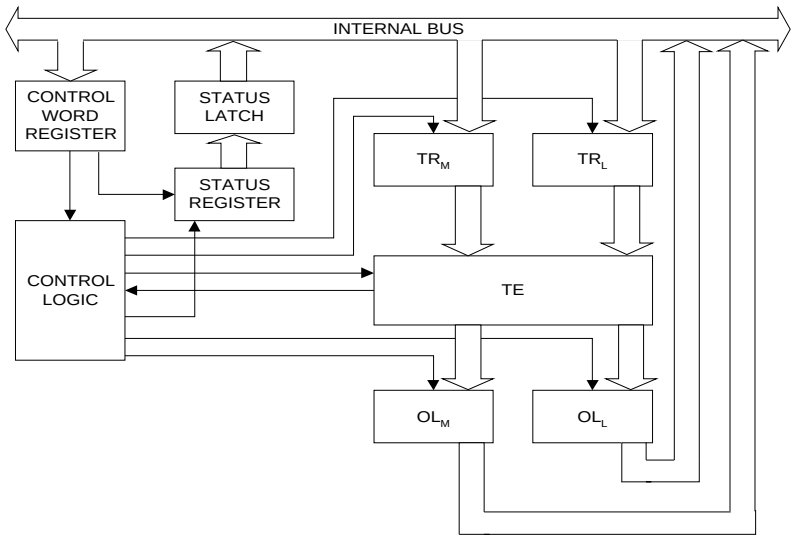


Figure 4-5 Internal Timer Diagram

Writing

The Timers are programmed by first writing a Control Word and then the initial count. The format of the Control Word is shown in Tables 4-2 through 4-6. All Control Words are written into the Control Word Register while the initial counts are written into the individual Timer registers. The format of the initial count is determined by the Control Word.

Table 4-2 Control Word Format

D7	D6	D5	D4	D3	D2	D1	D0
ST1	ST0	RW1	RW0	M2	M1	M0	BCD

Table 4-3 ST - Select Timer

ST1*	ST0*	Description
0	0	Select Timer 0
0	1	Select Timer 1
1	0	Select Timer 2
1	1	Read-Back Command (See Reading section on page 59)
*The ST bits specify which Timer (0, 1, or 2) the Control word refers to or whether this is a Read-Back command		

.

Table 4-4 RW - Read/Write

RW1*	RW0*	Description
0	0	Timer Latch Command (see Reading section)
0	1	Read / Write least significant byte only
1	0	Read / Write most significant byte only
1	1	Read / Write least significant byte first, then most significant
*The RW bits specify whether this is a Timer Latch command or the byte ordering of the Read / Write transaction.		

Table 4-5 M - Mode

M2*	M1*	M0*	Description
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5
* Only Mode 2 is described in this manual.			

Table 4-6 BCD

BCD*	Description
0	Binary Timer 16-bits
1	Binary Coded Decimal (BCD) Timer (4 Decades)
* The BCD bit specifies whether the Timer count value is in Binary or BCD.	

When programming the 82C54, only two rules need to be followed.

1. For each Timer, the Control Word must be written first.
2. The initial count must follow the format specified in the Control Word (least significant byte only, most significant byte only, or least significant byte then most significant byte). As long as these rules are adhered to, any programming sequence is acceptable.

Reading

There are two methods for reading the timers: the Timer Latch Command and the Read-Back Command.

Timer Latch Command

The Timer Latch Command allows the reading of a Timer 'on the fly' without affecting the timing in process.

Like a Control Word, the Timer Latch Command is written to the Control Word Register (I/O Address $503, see Table 4-1). The Select Timer bits (ST1, ST0, see Table 4-3) select one of the three timers while the Read / Write bits (RW1,RW0, see Table 4-4) select the Timer Latch Command, RW1 = 0 and RW0 = 0. The selected Timer's count is latched into the 0L registers at the time of the Timer Latch Command. The count is held in the 0L latches until it is read. Multiple Timer Latch Commands can be used to latch more than one Timer. Again, each Timer's count is held latched until it is read.

Read-Back Command

The Read-Back Command allows the user to view the Timer count, the Timer Mode, the current state of the OUT pin, and the Load Flag of the selected Timer. Like a Control Word, the Read-Back Command is written into the Control Word Register and has the format shown in Tables 4-7 and 4-8. The Command applies to the Timer(s) selected by setting the corresponding bits Cnt2, Cnt1, Cnt0 = 1.

Table 4-7 Read-Back Command Format

D7	D6	D5	D4	D3	D2	D1	D0
1	1	Count	Status	Cnt2	Cnt1	Cnt0	0

Table 4-8 Read-Back Command Description

Bit	Description
D5: $\overline{\text{Count}}$	Latch count of selected Timer(s)
D4: $\overline{\text{Status}}$	Latch status of selected Timer(s)
D3: Cnt2	Select Timer 2
D2: Cnt1	Select Timer 1
D1: Cnt0	Select Timer 0
D0	Reserved, must be 0

The Read-Back Command can be used to latch several Timer counts by setting the $\overline{\text{Count}}$ bit = 0 and selecting the Timers. This is the same as using multiple Timer Latch Commands. Again each Timer's latched count will be held until it is read.

The Read-Back command can also be used to latch the timer status by setting the $\overline{\text{Status}}$ bit = 0 and selecting the Timers. Status of a Timer is accessed by a read from that Timer (see Table 4-1 on page 55). If more than one Timer Status Read-Back command is issued without reading the status, all but the first is ignored.

The format of the Timer Status byte is shown in Tables 4-9 and 4-10.

Table 4-9 Status Byte

D7	D6	D5	D4	D3	D2	D1	D0
OUT	LOAD	RW1	RW0	M2	M1	M0	BCD

Table 4-10 Status Byte Description

Bit	Description
D7: OUT	Current state of Timers OUT pin
D6: LOAD	Count loaded into Timer
D5-D0	Timer Programmed Mode

Bit D7 contains the state of the Timers OUT pin. This allows viewing of the Timer's OUT pin via software.

Bit D6 indicates when the count written to the Timer is actually loaded into the Timer register. The exact time of the loading depends on the Mode the Timer is in and is defined in the "Mode Definitions" section. The count cannot be read from the Timer until it has been loaded. If a count is read before this time, the value read will not be the new count just written. Refer to Table 4-11.

Bits D5 through D0 contain the Timer's programmed mode exactly, bit for bit, like the Timer Control Words bits D5 through D0. See Table 4-2 on page 57.

Table 4-11 LOAD Bit Operation

Action	Causes
1. Write to the Control Word Register [1]	LOAD bit = 1
2. Write count to Timer [2]	LOAD bit = 1
3. New count loaded into Timer	LOAD bit = 0

[1]Only the Timer specified in the Control Word will have its LOAD bit set to 1. LOAD bits of other Timers are not affected.

[2]If the Timer is programmed for two byte counts (least significant then most significant), the LOAD bit will go to 1 when the second byte is written.

Both the count and status of the specified Timer(s) can be latched at the same time by setting both the $\overline{\text{Count}}$ bit (D5) and $\overline{\text{Status}}$ bit (D4) to zero (0) in the Read-Back command. If this technique is used, the first read operation of the Timer will return the status while the next one or two reads (depending on whether the Timer is programmed for one or two bytes) will return the count. Succeeding reads will return unlatched counts.

Mode Definitions

The VMIVME-7698 utilizes an 82C54 Timer / Counter for its Timers. 82C54 Timer / Counters can be programmed to function in six different modes (numbered Mode 0 through Mode 5). The VMIVME-7698 Timers are hardware configured to operate using Mode 2. Only Mode 2 is defined.

Mode 2 functions as a divide by N counter. Once a Control Word and an initial count are written to the Timer, the initial count is loaded on the next Clock cycle. When the count decrements to 1 an interrupt is generated. The Timer then reloads the initial count and the process repeats. This Mode is periodic. For an initial count of N, the sequence repeats every N CLK cycles. An initial count of 1 is illegal.

Writing a new count while the Timer is counting does not affect the current sequence. The new count will be loaded at the end of current sequence.

Flash Disk

The VMIVME-7698 features an on-board Flash mass storage system that allows the host computer to issue commands to read or write blocks to memory in a Flash memory array. This flash disk appears to the user as an intelligent ATA (IDE) disk drive with the same functionality and capabilities as a “rotating media” IDE hard drive.

Configuration

The flash disk resides on the VMIVME-7698 as the secondary IDE bus master device (the secondary IDE bus slave device is not assignable). The default setting is AUTO. This can be seen in the BIOS menus. From the Main BIOS menu select Secondary Master and press the Return key. The default type will be AUTO. Refer to Appendix C, Phoenix BIOS for additional details.

Figure 4-6 maps the configuration possibilities for a typical system consisting of the VMIVME-7698 with a resident flash disk, a hard drive attached to the Primary IDE interface, and a floppy drive attached to the floppy interface.

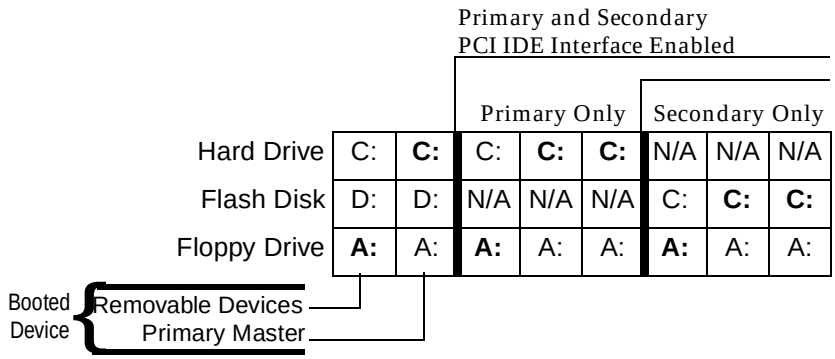


Figure 4-6 Typical System Configuration

The Primary and Secondary PCI IDE Interfaces are controlled (enabled or disabled) in the Advanced screen of the Phoenix BIOS. The boot sequence is determined by the order of the stack in the Boot menu.

Figure 4-6 identifies the drive letter assigned to each physical device and indicates in bold lettering the device booted from in each configuration, using devices that were bootable. The term bootable refers to a device on which an operating system has been installed or has been formatted as a system disk using MS-DOS.

Functionality

The flash disk performs identically to a standard IDE hard drive. Reads and writes to the device are performed using the same methods, utilizing DOS command line entries or the file managers resident in the chosen operating system.

Advanced Configuration

The previous discussion is based on using the IDE disk devices formatted as one large partition per device. Some applications may require the use of multiple partitions. The following discussion of partitions includes the special procedures that must be followed to allow the creation of multiple partitions on the VMIVME-7698 IDE disk devices (including the resident Flash Disk).

Partitions may be either a primary partition or an extended partition. An extended partition may be subdivided farther into logical partitions. Each device may have up to four main partitions, one of which may be an extended partition. However, if multiple primary partitions are created, only one partition may be active at one time. Data in the non-active partitions are not accessible.

Following the creation of the partitioning scheme, the partitions can be formatted to contain the desired file system.

As discussed earlier, a typical system consists of the VMIVME-7698 with its resident Flash Disk configured as the Secondary IDE device, a hard drive attached to the Primary IDE interface, and a floppy drive attached to the floppy interface.

Using this configuration, it may be desirable to have a logical device on either IDE device configured as a bootable device, allowing the selection of the boot sequence via the BIOS Features Setup screen. Using this capability, a user could have a system configured with multiple operating systems (OS's) that would then be selectable by assigning the IDE logical device as the boot device.

The DOS utility FDISK is commonly used to configure the partition structure on a hard drive. Other utility programs are available for performing this task. Partition Magic by PowerQuest is a popular and capable commercially available program. Comments that follow pertain to partitioning efforts using FDISK.

CAUTION: Deleting a partition will erase all the data previously held in that partition.

The Flash Disk will be configured as a single partition device as delivered from the factory. The following sample sequence illustrates a proven method for creating two partitions, with one as an active primary partition. Take note of the instructions to exit FDISK. This has been shown to be an important step in a successful partitioning effort.

1. Power up the VMIVME-7698, and enter CMOS Set-up.

2. Set Primary Master to **None**. Set Secondary Master to **Auto**. Set boot device to floppy.
3. Boot DOS from the floppy; verify, on the Bootup screen, that the Flask Disk is shown as Flash Disk 0.
4. Run FDISK.
5. Delete all current partitions (any data currently stored in the partitions will be lost).
6. Exit FDISK, this will cause a reboot, then run FDISK again.
7. Create a primary partition.
8. Create a extended partition and set-up a logical device for it.

NOTE: If only one partition is required its size will be equal to the total configurable memory available within the Flash Disk.

---

9. Set the Primary partition as an active partition.
10. Exit FDISK.

NOTE: If an operating system has been installed on the Flash Disk that modifies the Master Boot Record (MBR), then the following step is required to rewrite the MBR for DOS

---

11. Run FDISK /MBR.
12. Run FORMAT C: (use the /s option if you want the Flash Disk as a bootable DOS device.)
13. Format D: (This is only required if two partitions were created).
14. Reset the CPU and enter the CMOS set-up.
15. Set Primary Master to "AUTO."
16. Set boot device to desired boot source.

Drive letter assignments for a simple system were illustrated in Figure 4-6. Understanding the order the operating system assigns drive letters is necessary for these multiple partition configurations. The operating system assigns drive letter C: to the active primary partition on the first hard disk (the boot device). Drive D: is assigned to the first recognized primary partition on the next hard disk. The operating

system will continue to assign drive letters to the primary partitions in an alternating fashion between the two drives. Next, logical partitions will be assigned drive letters starting on the first hard drive lettering each logical device sequentially until they are all named, then doing the same sequential lettering of each logical partition on the second hard disk.

NOTE: Drive letter changes caused by adding a drive or changing the initial partitioning scheme may cause difficulties with an operating system installed prior to the changes. Plan your configuration prior to installing the operating system to minimize difficulties.

Watchdog Timer

The VMIVME-7698 uses a Dallas DS1384 Watchdog Timekeeping Controller as its Watchdog Timer. The device provides a Time of Day feature, a Watchdog Alarm and a Non-Volatile SRAM controller. The Time of Day feature found within the DS1384 device is explained in this section, but is not utilized by the VMIVME-7698. The actual Time of Day registers used by the VMIVME-7698 are located at the standard PC/AT I/O address. The Time of Day feature in the DS1384 Watchdog Timer is available for use by the user at their discretion. The Watchdog Timer provides a Watchdog alarm window and interval timing between 0.01 and 99.99 seconds.

The Non-Volatile SRAM is explained in the Battery Backed SRAM section of this manual.

The Watchdog alarm can, under software control, generate a VME SYSFAIL. Bit 8 of the System Command register (see Table 3-1) is used to enable this option. The System Command Register is 2 bytes wide located at memory address $D800E.

NOTE: The Watchdog Timer Interrupt output must be set to Level Mode (see Watchdog Command Register bit 4) to use this option.

In addition, the Watchdog Alarm is connected via a two state header (E8) to either or neither the CPU reset or the system NMI. The user can direct the Watchdog Alarm to reset the CPU if the jumper is set in the 2-4 position; to initiate a Non-Maskable Interrupt (NMI) if the jumper is in the 4-6 position; or neither if the jumper is completely removed.

Figure 4-7 shows a generalized block diagram of how the Watchdog Timer is used in the VMIVME-7698. The Watchdog Timer registers are memory-mapped in the bottom fourteen locations of battery-backed SRAM addresses $D8000 through $D800D. Table 4-12 shows the address, content, and the range of each Watchdog Register.

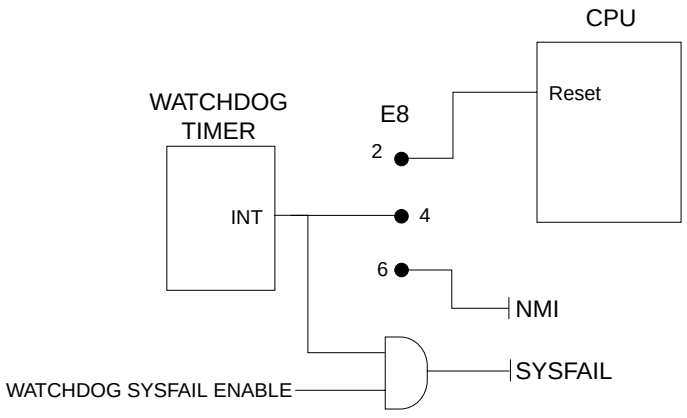


Figure 4-7 Watchdog Alarm Block

Table 4-12 Watchdog Registers

Register	Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Range
0	$D8000	0.1 Seconds (BCD)				0.01 Seconds (BCD)				00 - 99
1	$D8001	10 Seconds (BCD)				Seconds (BCD)				00 - 59
2	$D8002	10 Minutes (BCD)				Minutes (BCD)				00 - 59
3	$D8003	M	10 Minute Alarm (BCD)			Minute Alarm (BCD)				00 - 59
4	$D8004	0	12/24	AM/PM*	10 Hr	Hours (BCD)				
5	$D8005	M	12/24	AM/PM*	10 Hr	Hour Alarm (BCD)				
6	$D8006	0	0	0	0	Days (BCD)				01 - 07
7	$D8007	M	0	0	0	Day Alarm (BCD)				01 - 07
8	$D8008	0	0	10 Date (BCD)		Date (BCD)				01 - 31
9	$D8009	Eosc	1**	0	10 Mo	Months (BCD)				01 - 12
A	$D800A	10 Years (BCD)				Years (BCD)				00 - 99
B	$D800B	Te	Ipsw	Ibh/lo	Pu/lvl	Wam	Tdm	Waf	Tdf	
C	$D800C	0.1 Seconds (BCD)				0.01 Seconds (BCD)				00 - 99
D	$D800D	10 Seconds (BCD)				Seconds (BCD)				00 - 99

** In the 12 hour mode Bit 5 determines AM (0) or PM (1). In the 24 hour mode Bit 5 combines with Bit 4 to represent the 10 hour value.*
*** Bit 6 of Register 9 must be set to a 1. If set to a 0, an unused square wave will be generated in the circuit.*

Registers 0 through A are Clock, Calendar, Time of Day Registers.
Register B is the Command Register.
Registers C and D are Watchdog Alarm Registers.

The Watchdog Timer contains 14 registers which are 8-bits wide. These registers contain all of the Time of Day, Alarm, Watchdog, Control, and Data information. The Clock Calendar, Alarm, and Watchdog Registers have both external (user accessible) and internal memory locations containing copies of the data. The external memory locations are independent of the internal functions except that they are updated periodically by the transfer of the incremented internal values. Registers 0, 1, 2, 4, 6, 8, 9, and A contain Time of Day and Data information in Binary Coded Decimal (BCD). Registers 3, 5, and 7 contain the Time of Day Alarm information in BCD. The Command Register (Register B) contains data in binary. The Watchdog Alarm Registers are Registers C and D and information stored in these registers is in BCD.

Time of Day Registers

Registers 0, 1, 2, 4, 6, 8, 9, and A contain Time of Day data in BCD.

Register 0 contains two Time of Day values. Bits 3 - 0 contain the 0.01 Seconds value with a range of 0 to 9 in BCD while Bits 7 - 4 contain the 0.1 Seconds value with a range of 0 to 9 in BCD. This Register has a total range of 0.00 to 0.99 Seconds.

Register 1 contains two Time of Day values. Bits 3 - 0 contain the 1 Seconds value with a range of 0 to 9 in BCD while Bits 7 - 4 contain the 10 Seconds value with a range of 0 to 5 in BCD. This Register has a total range of 0.0 to 59.0 Seconds. Bit 7 of this register will always be zero regardless of what value is written to it.

Register 2 contains two Time of Day values. Bits 3 - 0 contain the 1 Minute value with a range of 0 to 9 in BCD, Bits 7 - 4 contain the 10 Minutes value with a range of 0 to 6 in BCD. This Register has a total range of 0 to 59 Minutes. Bit 7 of this register will always be zero regardless of what value is written to it.

Register 4 contains the Hours value of the Time of Day. The Hours can be represented in either 12- or 24-hour format depending on the state of Bit 6. When Bit 6 is set to a one (1) the format is 12-hour. When Bit 6 is set to a zero (0) the format is 24-hour. For the 12-hour format Bits 3 - 0 contain the 1-hour value with a range of 0 to 9 in BCD and Bit 4 contains the 10-hour value with a range of 0 to 1. In the 12-hour format Bit 5 is used as the AM / PM bit. For AM, Bit 5 is set to zero (0) and for PM, bit 5 is set to one (1). The total range of this register in the 12 hour format is 01 AM to 12 AM and 01 PM to 12 PM.

When Register 4 is in 24-hour format (Bit 6 is set to a zero (0)) Bits 3 - 0 contain the 1-hour value with a range of 0 to 9 in BCD, Bit 5 combines with 4 to represent the 10-hour value. The 10-hour range is from 0 to 2. The total range of register 4 in the 24-hour format is 00 to 23 hours. Bit 7 of register 4 will always be zero regardless of what value is written to it and regardless of format (12- or 24-hour).

Register 6 contains the Days value of the Time of Day. Bits 2 - 0 contain the Days value with a range of 1 to 7 in BCD.

Register 8 contains two Time of Day values. Bits 3 - 0 contain the Date value with a range of 0 to 9 in BCD while Bits 5 - 4 contain the 10 Date value with a range of 0 to 3. This Register has a total range of 01 to 31. Bits 7 - 6 of this register will always be zero regardless of what value is written to it.

Register 9 contains two Time of Day values. Bits 3 - 0 contain the Months value with a range of 0 to 9 in BCD while Bits 4 contain the 10 Date value with a range of 0 to 1. This Register has a total range of 01 to 12. Bit 5 will always be zero regardless of what value is written to it. Bit 6 is unused but must be set to a 1. Bit 7, $\overline{\text{Eosc}}$, is the clock oscillator enable bit. When this bit is set to a zero (0) the oscillator is internally enabled. When set to a one (1) the oscillator is internally disabled. The oscillator via this bit is usually turned on once during system initialization but can be toggled on and off at the user's discretion.

There are two techniques for reading the Time of Day from the Watchdog Timer. The first is to halt the external Time of Day registers from tracking the internal Time of Day registers by setting the Te bit (Bit 7 of the Command Register) to a logic zero (0), then reading the contents of the Time of Day registers. Using this technique eliminates the chance of the Time of Day changing while the read is taking place. At the end of the read, the Te bit is set to a logic one (1) allowing the external Time of Day registers to resume tracking the internal Time of Day Registers. No time is lost as the internal Time of Day Registers continue to keep time while the external Time of Day registers are halted. This is the recommended method.

The second technique for reading the Time of Day from the Watchdog Timer is to read the external Time of Day registers without halting the tracking of the internal Registers. This is not recommended as the registers may be updated while the reading is taking place, resulting in erroneous data being read.

Time of Day Alarm Registers

Registers 3, 5, and 7 are the Time of Day Alarm registers and are formatted similar to Register 2, 4, and 6 respectively. Bit 7 of registers 3, 5, and 7 is a mask bit. The mask bits, when active (logic one (1)), disable the use of the particular Time of Day Alarm register in the determination of the Time of Day Alarm (see Table 4-13). When all the mask bits are low (0) an alarm will occur when Registers 2, 4, and 6 match the values found in registers 3, 5, and 7. When Register 7's mask bit is set to a logic one (1), register 6 will be disregarded in the determination of the Time of Day alarm and an alarm will occur everyday. When registers 7 and 5's mask bit is set to a logic one (1), Register 6 and 4 will be disregarded in the determination of the Time of Day alarm and an alarm will occur every hour. When Registers 7, 5 and 3's mask bit is set to a logic one (1), Register 6, 4, and 2 will be disregarded in the determination of the Time of Day alarm and an alarm will occur every minute (when register 1's seconds step from 59 to 00).

Table 4-13 Time of Day Alarm Registers

Register			Comment
Minutes	Hours	Days	
1	1	1	Alarm once per minute
0	1	1	Alarm when minutes match
0	0	1	Alarm when hours and minutes match
0	0	0	Alarm when hours, minutes, and days match

The Time of Day Alarm registers are read and written to in the same format as the Time of Day registers. The Time of Day alarm flag and interrupt are cleared when the alarm registers are read or written.

Watchdog Alarm Registers

Register C contains two Watchdog alarm values. Bits 3 - 0 contain the 0.01 Seconds value with a range of 0 to 9 in BCD while Bits 7 - 4 contain the 0.1 Seconds value with a range of 0 to 9 in BCD. This Register has a total range of 0.00 to 0.99 Seconds.

Register D contains two Watchdog Alarm values. Bits 3 - 0 contain the 1 Second value with a range of 0 to 9 in BCD while Bits 7 - 4 contain the 10 Seconds value with a range of 0 to 9 in BCD. This Register has a total range of 00.0 to 99.0 Seconds.

The Watchdog Alarm Registers can be read or written in any order. When a new value is entered or the Watchdog registers are read, the Watchdog Timer will start counting down from the entered value. When zero is reached the Watchdog Interrupt Output will go active. If jumper J30 is loaded, the CPU will reset to a known state. Refer to Figure 4-7. The Watchdog Timer count is reinitialized back to the entered value, the Watchdog flag bit is cleared, and the Watchdog interrupt output is cleared every time either of the registers is accessed. Periodic accesses to the Watchdog Timer will prevent the Watchdog Alarm from occurring. If access does not occur, the alarm will be repetitive. The Watchdog Alarm Register always reads the entered value. The actual countdown value is internal and not accessible to the user. Writing zeroes to Registers C and D will disable the Watchdog Alarm feature.

Command Register

Register B is the Command Register. Within this register are mask bits, control bits, and flag bits. The following paragraphs describe each bit.

Te - Bit 7 Transfer Enable - This bit enables and disables the tracking of data between the internal and external registers. When set to a logic zero (0), tracking is disabled that is the data in the external register is frozen. When set to a logic one (1), tracking is enabled. This bit must be set to a logic one (1) to allow the external register to be updated.

Ipsw - Bit 6 Interrupt Switch - This bit toggles the Interrupt Output between the Time of Day Alarm and the Watchdog Alarm. When set to a logic zero (0), the Interrupt Output is from the Watchdog Alarm. When set to a logic one (1), the Interrupt Output is from the Time of Day Alarm.

Ibh/lo - Bit 5 Reserved - This bit should be set to a logic low (0).

Pu/lvl - Bit 4 Interrupt Pulse Mode or Level Mode - This bit determines whether the Interrupt Output will output as a pulse or a level. When set to a logic zero (0), Interrupt Output will be a level. When set to a logic one (1), Interrupt Output will be a pulse. In pulse mode the Interrupt Output will sink current for a minimum of 3 ms. This bit should be set to a logic one (1).

Wam - Bit 3 Watchdog Alarm Mask - Enables / Disables the Watchdog Alarm to Interrupt Output when Ipsw (Bit 6, Interrupt Switch) is set to logic zero (0). When set to a logic zero (0), Watchdog Alarm Interrupt Output will be enabled. When set to a logic one (1), Watchdog Alarm Interrupt Output will be disabled.

Tdm - Bit 2 Time of Day Alarm Mask - Enables / Disables the Time of Day Alarm to Interrupt Output when Ipsw (see Bit 6, Interrupt Switch) is set to logic one (1). When set to a logic zero (0), Time of Day Alarm Interrupt Output will be enabled. When set to a logic one (1), Time of Day Alarm Interrupt Output will be disabled.

Waf - Bit 1 Watchdog Alarm Flag - This is a read-only bit set to a logic one (1) when a Watchdog Alarm Interrupt occurs. This bit is reset when any of the Watchdog Alarm registers are accessed. When the Interrupt Output is set to Pulse Mode (see Bit 4, Interrupt Pulse Mode or Level Mode), the flag will be set to a logic one (1) only when the Interrupt Output is active.

Tdf - Bit 0 Time of Day Alarm Flag - This is a read-only bit set to a logic one (1) when a Time of Day Alarm Interrupt occurs. This bit is reset when any of the Time of Day Alarm registers are accessed. When the Interrupt Output is set to Pulse Mode (see Bit 4, Interrupt Pulse Mode or Level Mode), the flag will be set to a logic one (1) only when the Interrupt Output is active.

Battery Backed SRAM

The VMIVME-7698 includes 32 K byte of battery-backed SRAM addressed at $D8018 to $DFFFF. The lower 24 bytes, $D8000 to $D8017, are dedicated to the Watchdog Timer, the System Command Register, the VME BERR Address Register, the VME BERR Address Modifier Register and the Board ID Register and are unavailable for SRAM use. See the Watchdog Timer section. The battery-backed SRAM can be accessed by the CPU at anytime, and can be used to store system data that must not be lost during power-off conditions.

CHAPTER

5

Maintenance

If a VMIC product malfunctions, please verify the following:

1. Software resident on the product
2. System configuration
3. Electrical connections
4. Jumper or configuration options
5. Boards are fully inserted into their proper connector location
6. Connector pins are clean and free from contamination
7. No components or adjacent boards were disturbed when inserting or removing the board from the chassis
8. Quality of cables and I/O connections

If products must be returned, obtain a RMA (Return Material Authorization) by contacting VMIC Customer Service. **This RMA must be obtained prior to any return**.

VMIC Customer Service is available at: 1-800-240-7782.
Or E-mail us at customer.service@vmic.com

Maintenance Prints

User level repairs are not recommended. The drawings and diagrams in this manual are for reference purposes only.

Appendix

A

Connector Pinouts

Contents

Introduction

The VMIVME-7698 PC / AT-Compatible VMEbus Controller has several connectors for its I / O ports. Figure A-1 shows the locations of the connectors on the VMIVME-7698. Wherever possible, the VMIVME-7698 uses connectors and pinouts typical for any desktop PC. This ensures maximum compatibility with a variety of systems.

Connector diagrams in this appendix are generally shown in a natural orientation with the controller board mounted in a VMEbus chassis.

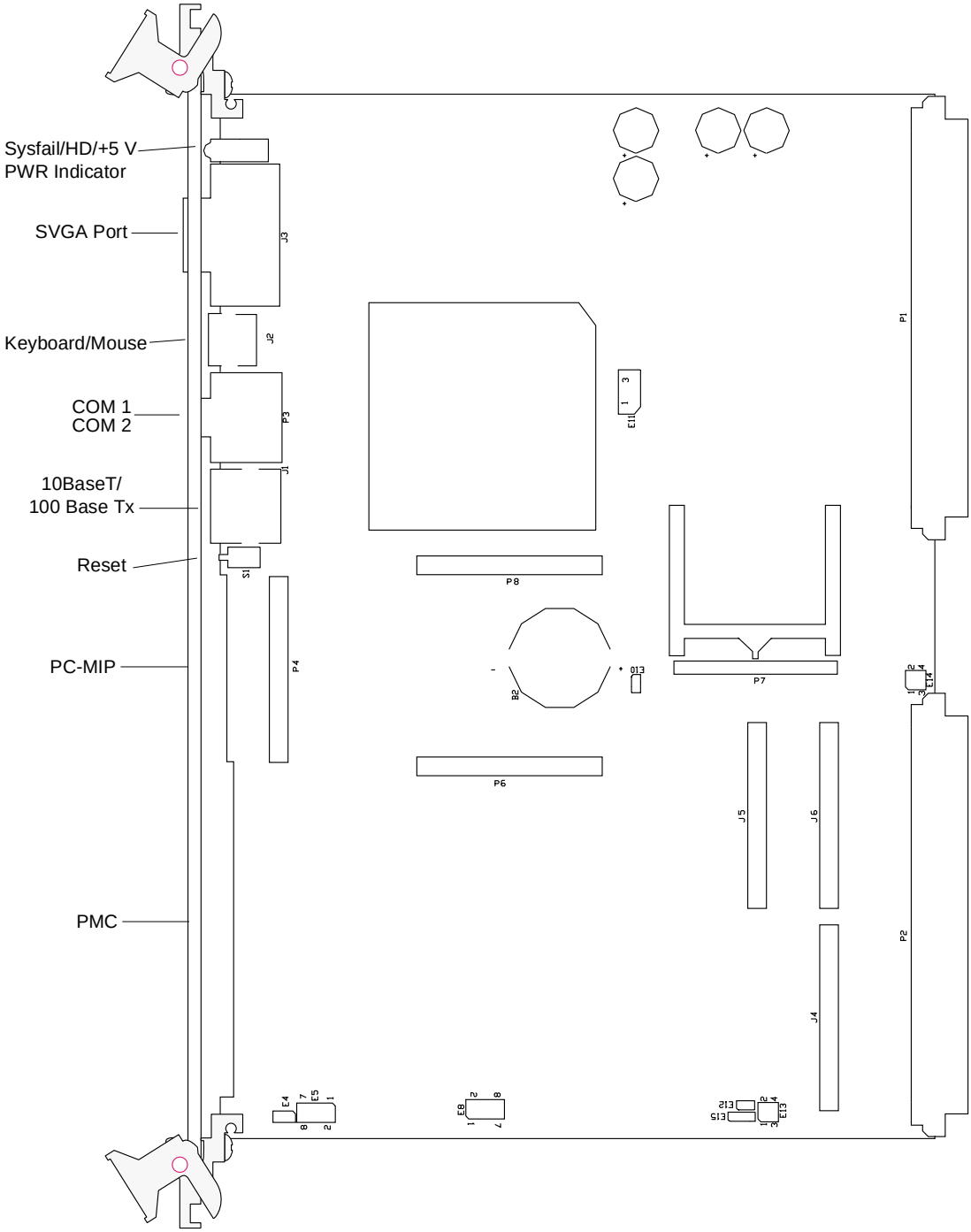
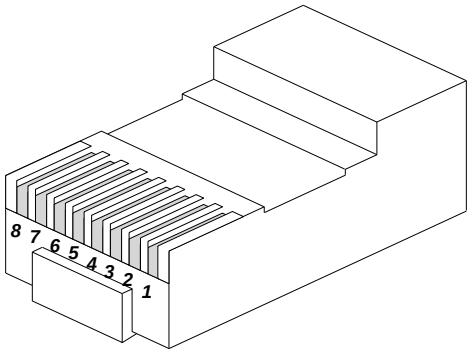


Figure A-1 VMIVME-7698 Connector and Jumper Locations

Ethernet Connector Pinout

The pinout diagram for the Ethernet 10BaseT / 100BaseTx and connectors are shown in Figure A-2.



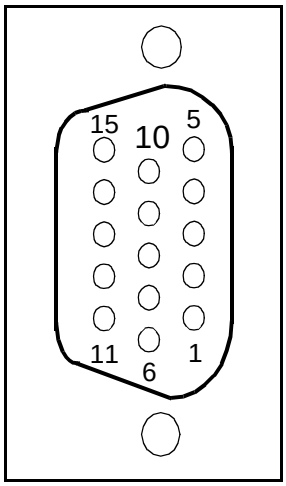
10BaseT/100BaseTx

ETHERNET CONNECTOR 10BaseT/100BaseTx		
PIN	**Signal Name**	
1	TD+	Transmit Data
2	TD-	Transmit Data
3	RD+	Receive Data
4	TX_CT_OUT	Transmit Center Tap Out
5	TX_CT_OUT	Transmit Center Tap Out
6	RD-	Receive Data
7	RX_CT_OUT	Receive Center Tap Out
8	RX_CT_OUT	Receive Center Tap Out

Figure A-2 Ethernet Connector Pinout

Video Connector Pinout

The video port uses a standard high-density D15 SVGA connector. Figure A-3 illustrates the pinout.

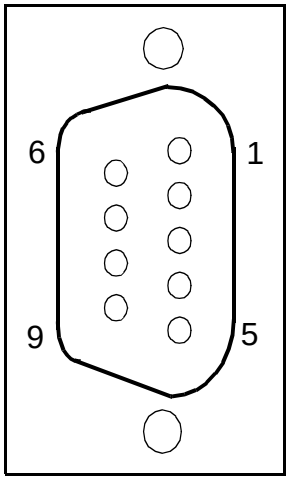


VIDEO CONNECTOR		
PIN	**DIRECTION**	**FUNCTION**
1	Out	Red
2	Out	Green
3	Out	Blue
4		Reserved
5		Ground
6		Ground
7		Ground
8		Ground
9		Reserved
10		Ground
11		Reserved
12		Reserved
13	Out	Horizontal Sync
14	Out	Vertical Sync
15		Reserved
Shield		Chassis Ground

Figure A-3 Video Connector Pinout

Serial Connector Pinout

Each standard RS-232 serial port connector is a Microminiature D9 male as shown in the drawing in Figure A-4. The cable length for these serial ports complies with the RS-232C recommended standard of 50 feet, maximum, in order to meet requirements of EN61000-4-5 for CE mark. Adapters to connect standard D9 serial peripherals to the board are available. Please refer to the product specification sheet for ordering information.

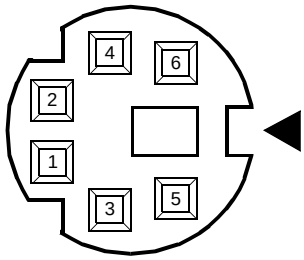


COM 1 and COM 2 SERIAL PORT CONNECTORS			
D9 PIN	**DIR**	**RS-232 SIGNAL**	**FUNCTION**
1	In	DCD	Data Carrier Detect
2	In	RX	Receive Data
3	Out	TX	Transmit Data
4	Out	DTR	Data Terminal Ready
5		GND	Signal Ground
6	In	DSR	Data Set Ready
7	Out	RTS	Request to Send
8	In	CTS	Clear to Send
9	In	RI	Ring Indicator
Shield			Chassis Ground

Figure A-4 Serial Connector Pinouts

Keyboard/Mouse Connector Pinout

The keyboard / Mouse connector is a standard 6-pin female mini-DIN PS / 2 connector as shown in Figure A-5. The Keyboard / Mouse Y-cable connects to the Keyboard / Mouse connector on the VMIVME-7698 and provides a separate connector for both Keyboard and Mouse. The pinout of these connectors are shown in Table A-1.



KEYBOARD/Mouse* CONNECTOR		
PIN	**DIR**	**FUNCTION**
1	In / Out	Mouse Data
2	In / Out	Keyboard Data
3		Ground
4		+5 V
5	Out	Mouse Clock
6	Out	Keyboard Clock
Shield		Chassis Ground

An adapter cable is included with the VMIVME-7698 to separate the Keyboard / Mouse Connector.

Figure A-5 Keyboard Connector Pinout

Table A-1 Keyboard/Mouse Y Splitter Cable

Keyboard			Mouse		
Pin	Dir	Function	Pin	Dir	Function
1	In/Out	Keyboard Data	1	In/Out	Mouse Data
2		Unused	2		Unused
3		Ground	3		Ground
4		+5V	4		+5V
5	Out	Keyboard Clock	5	Out	Mouse Clock
6		Unused	6		Unused
Shield		Chassis Ground	Shield		Chassis Ground

VMEbus Connector Pinout

Figure A-6 shows the location of the VMEbus P1 and P2 connectors and their orientation. Table A-2 and Table A-3 shows the pin assignments for the VMEbus connectors.

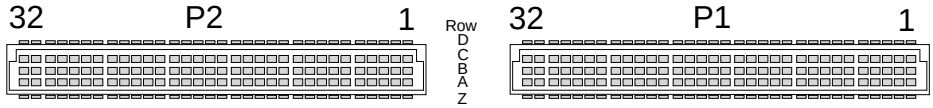


Figure A-6 VME64 Connector Diagram

Table A-2 P1 - VME64 Connector

A	Signal	Description	B	Signal	Description	C	Signal	Description
1	VME_D[0]	VME Data Bus Line 0	1	VME_BBSY-	Bus Busy	1	VME_D[8]	VME Data Bus Line 8
2	VME_D[1]	VME Data Bus Line 1	2	VME_BCLR-	Bus Clear	2	VME_D[9]	VME Data Bus Line 9
3	VME_D[2]	VME Data Bus Line 2	3	VME_ACFAIL-	Power Fail	3	VME_D[10]	VME Data Bus Line 10
4	VME_D[3]	VME Data Bus Line 3	4	VME_BG0IN-	Bus Grant 0 In	4	VME_D[11]	VME Data Bus Line 11
5	VME_D[4]	VME Data Bus Line 4	5	VME_BG0OUT-	Bus Grant 0 Out	5	VME_D[12]	VME Data Bus Line 12
6	VME_D[5]	VME Data Bus Line 5	6	VME_BG1IN-	Bus Grant 1 In	6	VME_D[13]	VME Data Bus Line 13
7	VME_D[6]	VME Data Bus Line 6	7	VME_BG1OUT-	Bus Grant 1 Out	7	VME_D[14]	VME Data Bus Line 14
8	VME_D[7]	VME Data Bus Line 7	8	VME_BG2IN-	Bus Grant 2 In	8	VME_D[15]	VME Data Bus Line 15
9	GND	Ground	9	VME_BG2OUT-	Bus Grant 2 Out	9	GND	Ground
10	VME_SYSCLK	VME Bus System Clock	10	VME_BG3IN-	Bus Grant 3 In	10	VME_SYSFAIL-	System Failure
11	GND	Ground	11	VME_BG3OUT-	Bus Grant 3 Out	11	VME_BERR-	Bus Error
12	VME_DS1-	Data Strobe 1	12	VME_BR0-	Bus Request 0	12	VME_SYSRST-	System Reset
13	VME_DS0-	Data Strobed	13	VME_BR1-	Bus Request 1	13	VME_LWORD-	Long Word
14	VME_WRITE-	Write Enable	14	VME_BR2-	Bus Request 2	14	VME_AM5	Address Modifier 5
15	GND	Ground	15	VME_BR3-	Bus Request 3	15	VME_A[23]	VME Address Line 23
16	VME_DTACK-	Data Acknowledge	16	VME_AM0	Address Modifier 0	16	VME_A[22]	VME Address Line 22
17	GND	Ground	17	VME_AM1	Address Modifier 1	17	VME_A[21]	VME Address Line 21
18	VME_AS-	Address Strobe	18	VME_AM2	Address Modifier 2	18	VME_A[20]	VME Address Line 20
19	GND	Ground	19	VME_AM3	Address Modifier 3	19	VME_A[19]	VME Address Line 19
20	VME_IACK-	VME IRQ Acknowledge	20	GND	Ground	20	VME_A[18]	VME Address Line 18
21	VME_IACKIN-	Interrupt Acknowledge	21	N.C.	No Connect	21	VME_A[17]	VME Address Line 17
22	VME_IACKOUT-	Int Ack Out	22	N.C.	No Connect	22	VME_A[16]	VME Address Line 16
23	VME_AM4	Address Modifier 4	23	GND	Ground	23	VME_A[15]	VME Address Line 15
24	VME_A[7]	VME Address Line 7	24	VME_IRQ[7]-	IRQ Line 7	24	VME_A[14]	VME Address Line 14
25	VME_A[6]	VME Address Line 6	25	VME_IRQ[6]-	IRQ Line 6	25	VME_A[13]	VME Address Line 13
26	VME_A[5]	VME Address Line 5	26	VME_IRQ[5]-	IRQ Line 5	26	VME_A[12]	VME Address Line 12
27	VME_A[4]	VME Address Line 4	27	VME_IRQ[4]-	IRQ Line 4	27	VME_A[11]	VME Address Line 11
28	VME_A[3]	VME Address Line 3	28	VME_IRQ[3]-	IRQ Line 3	28	VME_A[10]	VME Address Line 10
29	VME_A[2]	VME Address Line 2	29	VME_IRQ[2]-	IRQ Line 2	29	VME_A[9]	VME Address Line 9
30	VME_A[1]	VME Address Line 1	30	VME_IRQ[1]-	IRQ Line 1	30	VME_A[8]	VME Address Line 8
31	-12V	-12V Power	31	N.C.	No Connect	31	+12V	+12V Power
32	+5V	Power	32	+5V	Power	32	+5V	Power

Table A-2 P1 - VME64 Connector (Continued)

D	Signal	Description	Z	Signal	Description
1	N.C.	No Connect	1	N.C.	No Connect
2	N.C.	No Connect	2	N.C.	No Connect
3	N.C.	No Connect	3	N.C.	No Connect
4	N.C.	No Connect	4	N.C.	No Connect
5	N.C.	No Connect	5	N.C.	No Connect
6	N.C.	No Connect	6	N.C.	No Connect
7	N.C.	No Connect	7	N.C.	No Connect
8	N.C.	No Connect	8	N.C.	No Connect
9	GA[5]	Geographic Address 5	9	N.C.	No Connect
10	GA[0]	Geographic Address 0	10	N.C.	No Connect
11	GA[1]	Geographic Address 1	11	N.C.	No Connect
12	N.C.	No Connect	12	N.C.	No Connect
13	GA[2]	Geographic Address 2	13	N.C.	No Connect
14	N.C.	No Connect	14	N.C.	No Connect
15	GA[3]	Geographic Address 3	15	N.C.	No Connect
16	N.C.	No Connect	16	N.C.	No Connect
17	GA[4]	Geographic Address 4	17	N.C.	No Connect
18	N.C.	No Connect	18	N.C.	No Connect
19	N.C.	No Connect	19	N.C.	No Connect
20	N.C.	No Connect	20	N.C.	No Connect
21	N.C.	No Connect	21	N.C.	No Connect
22	N.C.	No Connect	22	N.C.	No Connect
23	N.C.	No Connect	23	N.C.	No Connect
24	N.C.	No Connect	24	N.C.	No Connect
25	N.C.	No Connect	25	N.C.	No Connect
26	N.C.	No Connect	26	N.C.	No Connect
27	N.C.	No Connect	27	N.C.	No Connect
28	N.C.	No Connect	28	N.C.	No Connect
29	N.C.	No Connect	29	N.C.	No Connect
30	N.C.	No Connect	30	N.C.	No Connect
31	N.C.	No Connect	31	N.C.	No Connect
32	N.C.	No Connect	32	N.C.	No Connect

Table A-3 P2 - VME64 Connector

A	Signal	Description	B	Signal	Description	C	Signal	Description
1	GND	Ground	1	+5V	Power	1	IDERST#	IDE Control
2	IDED[8]	IDE Data Line 8	2	GND	Ground	2	IDED[7]	IDE Data Line 7
3	IDED[9]	IDE Data Line 9	3	N.C.	No Connect	3	IDED[6]	IDE Data Line 6
4	IDED[10]	IDE Data Line 10	4	VME_A[24]	VME Address Line 24	4	IDED[5]	IDE Data Line 5
5	IDED[11]	IDE Data Line 11	5	VME_A[25]	VME Address Line 25	5	IDED[4]	IDE Data Line 4
6	IDED[12]	IDE Data Line 12	6	VME_A[26]	VME Address Line 26	6	IDED[3]	IDE Data Line 3
7	IDED[13]	IDE Data Line 13	7	VME_A[27]	VME Address Line 27	7	IDED[2]	IDE Data Line 2
8	IDED[14]	IDE Data Line 14	8	VME_A[28]	VME Address Line 28	8	IDED[1]	IDE Data Line 1
9	IDED[15]	IDE Data Line 15	9	VME_A[29]	VME Address Line 29	9	IDED[0]	IDE Data Line 0
10	N.C.	No Connect	10	VME_A[30]	VME Address Line 30	10	IOCSI.64#	IDE Control
11	IDELIOW0-	IDE Control	11	VME_A[31]	VME Address Line 31	11	GND	Ground
12	IDELIOR0-	IDE Control	12	GND	Ground	12	GND	Ground
13	IDERDY	IDE Control	13	+5V	Power	13	GND	Ground
14	GND	Ground	14	VME_D[16]	VME Data Line 16	14	IDESELA	IDE Control
15	GND	Ground	15	VME_D[17]	VME Data Line 17	15	N.C.	No Connect
16	GND	Ground	16	VME_D[18]	VME Data Line 18	16	N.C.	No Connect
17	A1	IDE Control	17	VME_D[19]	VME Data Line 19	17	A2	IDE Control
18	CSOP	IDE Control	18	VME_D[20]	VME Data Line 20	18	A0	IDE Control
19	HD_ACTA-	IDE Control	19	VME_D[21]	VME Data Line 21	19	CS1P	IDE Control
20	DRATE0	Floppy Control	20	VME_D[22]	VME Data Line 22	20	DENSEL	Floppy Control
21	GND	Ground	21	VME_D[23]	VME Data Line 23	21	INDEX-	Floppy Control
22	DR1-	Floppy Control	22	GND	Ground	22	MTR0-	Floppy Control
23	GND	Ground	23	VME_D[24]	VME Data Line 24	23	DR0-	Floppy Control
24	GND	Ground	24	VME_D[25]	VME Data Line 25	24	MTR1-	Floppy Control
25	GND	Ground	25	VME_D[26]	VME Data Line 26	25	STEP-	Floppy Control
26	GND	Ground	26	VME_D[27]	VME Data Line 27	26	WDATA-	Floppy Control
27	GND	Ground	27	VME_D[28]	VME Data Line 28	27	TRK0-	Floppy Control
28	GND	Ground	28	VME_D[29]	VME Data Line 29	28	RDATA-	Floppy Control
29	DSKCHG	Floppy Control	29	VME_D[30]	VME Data Line 30	29	HDSEL	Floppy Control
30	GND	Ground	30	VME_D[31]	VME Data Line 31	30	DIR-	Floppy Control
31	+5V	Power	31	GND	Ground	31	WGATE-	Floppy Control
32	+5V	Power	32	+5V	Power	32	WP-	Floppy Control

Table A-3 P2 - VME64 Connector (Continued)

D	Signal	Description	Z	Signal	Description
1	CONN[1]	P8-2	1	CONN[2]	P8-2
2	CONN[3]	P8-3	2	GND	Ground
3	CONN[4]	P8-4	3	CONN[5]	P8-5
4	CONN[6]	P8-6	4	GND	Ground
5	CONN[7]	P8-7	5	CONN[8]	P8-8
6	CONN[9]	P8-9	6	GND	Ground
7	CONN[10]	P8-10	7	CONN[11]	P8-11
8	CONN[12]	P8-12	8	GND	Ground
9	CONN[13]	P8-13	9	CONN[14]	P8-14
10	CONN[15]	P8-15	10	GND	Ground
11	CONN[16]	P8-16	11	CONN[17]	P8-17
12	CONN[18]	P8-18	12	GND	Ground
13	CONN[19]	P8-19	13	CONN[20]	P8-20
14	CONN[21]	P8-21	14	GND	Ground
15	CONN[22]	P8-22	15	CONN[23]	P8-23
16	CONN[24]	P8-24	16	GND	Ground
17	CONN[25]	P8-25	17	CONN[26]	P8-26
18	CONN[27]	P8-27	18	GND	Ground
19	CONN[28]	P8-28	19	CONN[29]	P8-29
20	CONN[30]	P8-30	20	GND	Ground
21	CONN[31]	P8-31	21	CONN[32]	P8-32
22	CONN[33]	P8-33	22	GND	Ground
23	CONN[34]	P8-34	23	CONN[35]	P8-35
24	CONN[36]	P8-36	24	GND	Ground
25	CONN[37]	P8-37	25	CONN[38]	P8-38
26	CONN[39]	P8-39	26	GND	Ground
27	CONN[40]	P8-40	27	CONN[41]	P8-41
28	CONN[42]	P8-42	28	GND	Ground
29	CONN[43]	P8-43	29	CONN[44]	P8-44
30	CONN[45]	P8-45	30	GND	Ground
31	GND	Ground	31	CONN[46]	P8-46
32	N.C.	No Connect	32	GND	Ground

Appendix

B

System Driver Software

Contents

Introduction

The VMIVME-7698 provides high-performance video and Local Area Network (LAN) access by means of on-board PCI-based adapters and associated software drivers. The PCI-based video adapter used on the VMIVME-7698 is the Intel 69000. High-performance LAN operation, including 10BaseT and 100BaseTx, is provided by the Intel 82559 Ethernet controller chip.

To optimize performance of each of these PCI-based subsystems, the VMIVME-7698 is provided with software drivers compatible with DOS, Windows 95, and Windows NT operating systems. The following paragraphs provide instructions for loading and installing the adapter software.

Driver Software Installation

In order to properly use the Video and LAN adapters of the VMIVME-7698, the user must install the driver software located on the distribution diskettes provided with the unit. Detailed instructions for installation of the drivers during installation of Windows 95 or Windows NT (Versions 4.0) operating systems are described in the following sections.

Windows 95

1. Format the hard drive with MS-DOS.
2. Begin installation of Windows 95, following the instructions provided by the Windows 95 manual.
3. At the **Windows 95 Setup Wizard Screen**, choose **Typical** under **Setup Options** and click **Next**.
4. At the **Analyzing Your Computer** screen, place an **X** in the box for **Network Adapter**, then click **Next**.
5. Under the **Windows Components Screen**, select **Install The Most Common Components** and then click **Next**.
6. Continue with the installation until Windows 95 is completely installed and has rebooted.
7. When Windows 95 is restarted for the first time after installation, the **New Hardware Found** dialog window appears. Select **Driver From Disk Provided By Manufacturer**, click **OK**.
8. When the **Install From Disk** window appears, insert VMIC Driver Disk 320-500037-005 into drive A: and ensure **A:** is in the **Copy Manufacturer's Files From** box. **Click OK**. Follow the on screen prompts to complete the installation.
9. From **Control Panel**, double-click the **Network** icon.
10. Under **Network**, click on **File And Print Sharing** and choose the appropriate items for your system, click **OK**.
11. At the **Network** window, click **OK**. When prompted, insert the diskettes needed to complete the network installation, if required.
12. When the system prompts you to restart your computer, click on **Yes** for the network settings to take effect.

Video Driver

Follow the instructions below to install the video driver for Windows 95.

Part 1

1. Insert the Windows 95 drivers diskette into drive **A:**.
2. From the **Start** menu, select **Run**.
3. Type **`A:\W95500`**.
4. When the **Welcome** prompts, click on **Next**.
5. If you agree with the license agreement click **Yes**.
6. Click on 'Finish' to complete the installation and restart the computer.

Part 2

Verify that Windows 95 is actually using the CHIPS driver.

1. Open the Windows 95 control panel.
2. Double click the **Display** icon.
3. Click on the **Settings** tab.
4. Check Adapter/Driver Information:
   - For Win95a, select **Change Display Type**.
   - For Win95b, select **Advanced Properties**.

If the driver is installed and operating properly, the Chips and Technologies device is specified in the **Adapter Type** box.

Windows NT (Version 4.0)

Windows NT 4.0 includes drivers for the on-board LAN, and video adapters. The following steps are required to configure the LAN for operation.

1. Follow the normal Windows NT 4.0 installation until you reach the **Windows NT Workstation Setup** window which states that **Windows NT Needs To Know How This Computer Should Participate On A Network**.
2. Place a dot next to **This Computer Will Participate On A Network**.
3. Place a check mark next to **Wired To The Network** and click on **Next**.
4. In the next screen, click the **Select From List** button.
5. Click the **Have Disk** button.
6. Insert disk **320-500037-006** into drive **A:**.
7. Click **OK**.
8. In the **Select OEM Option**, choose **Intel Pro Adapter**, then click **OK**.
9. Select the above entry on the displayed list, click **Next**.
10. Select the **NetBEUI Protocol (only)**, click **Next**.
11. Click **Next** to install selected components.
12. Click **Next** to start the network connection.
13. Step through the remaining screens, providing the data pertinent to your network.
14. Continue through the setup procedure until the **Detected Display** window appears, click **OK** to continue.
15. In the **Display Properties** window, click on **Test**.

NOTE: Windows NT 4.0 does not allow the selection of the Intel video drivers during initial setup.

If the display test is successful, click **OK** to continue. If the display test is not successful, you may have to adjust the display parameter to find a functional setting, for example a lower resolution or lower number of colors.

16. Continue with the procedure to the **Windows NT Setup** window. Click **Restart Computer**.
17. When the computer reboots, double-click **My Computer** window.
18. Double-click the **Control Panel** icon in the **My Computer** window.
19. Double-click the **Display** icon in the **Control Panel**.

20. Select the **Settings** tab in the **Display Properties** window, then click the **Display Type** button.
21. In the **Display Type** window, click **Change**.
22. In the **Change Display** window, click **Have Disk**.
23. Insert disk **320-500037-003** into drive **A:**
24. Click **OK**.
25. **Intel 69000** will be displayed in the **Change Display** window. Click **OK**.
26. Proceed as directed, removing the driver disk from the floppy drive, and the computer will restart to activate the new settings. When the system reboots, the **Invalid Display Settings** screen will be displayed. Click **OK**.
27. On the **Display Properties** screen click on **Settings**, then click **Test**.
28. The **Testing Mode** screen will be displayed. Click **Ok**. If the bitmap test image is displayed correctly, click **OK**.

The unit should now be configured for operation under Windows NT 4.0.

Appendix

C

Phoenix BIOS

Contents

Introduction

The VMIVME-7698 utilizes the BIOS (Basic Input/Output System) in the same manner as other PC/AT compatible computers. This appendix describes the menus and options associated with the VMIVME-7698 BIOS.

System BIOS Setup Utility

During system bootup, press the F2 key to access the Phoenix BIOS Main screen. From this screen, the user can select any section of the Phoenix (system) BIOS for configuration, such as floppy drive configuration or system memory.

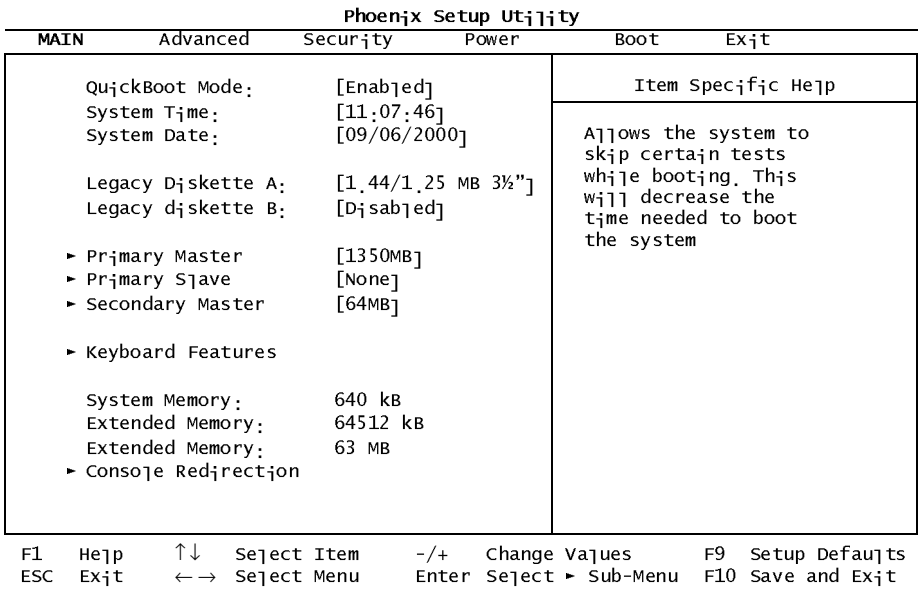
The parameters shown throughout this section are the default values.

Help Window

The help window on the right side of each menu displays the help text for the currently selected field. It updates as you move the cursor to each field. Pressing F1 or Alt- H on any menu brings up the General Help window that describes the legend keys and their alternates. The scroll bar on the right of any window indicates that there is more than one page of information in the window. Use PgUp and PgDn to display all the pages. Pressing Home and End displays the first and last page. Pressing Enter displays each page and then exits the window. Press Esc to exit the current window.

Main Menu

The Main menu allows the user to select QuickBoot, set the system clock and calendar, record disk drive parameters, and set selected functions for the keyboard.



QuickBoot

When enabled, certain checks normally performed during the POST are omitted, decreasing the time required to run the POST. The default is Enabled.

Setting The Time

The time format is based on the 24-hour military-time clock. For example, 1 PM is 13:00:00. Press the left or right arrow key to move the cursor to the desired field (hour, minute, seconds). Press the PgUp or PgDn key to step through the available choices, or type in the information.

Setting The Date

Press the left or right arrow key to move the cursor to the desired field (month, day, year). Press the PgUp or PgDn key to step through the available choices, or type in the information.

Legacy Diskette

Floppy Drive A

The VMIVME-7698 supports one floppy disk drive. The options are:

- Disabled — No diskette drive installed
- 360K, 5.25 in — 5-1/4 inch PC-type standard drive; 360 kilobyte capacity
- 1.2M, 5.25 in — 5-1/4 inch AT-type high-density drive; 1.2 megabyte capacity
- 720K, 3.5 in — 3-1/2 double-sided drive; 720 kilobyte capacity
- 1.44M, 3.5 in — 3-1/2 inch double-sided drive; 1.44 megabyte capacity
- 2.88M, 3.5 in — 3-1/2 inch double-sided drive; 2.88 megabyte capacity

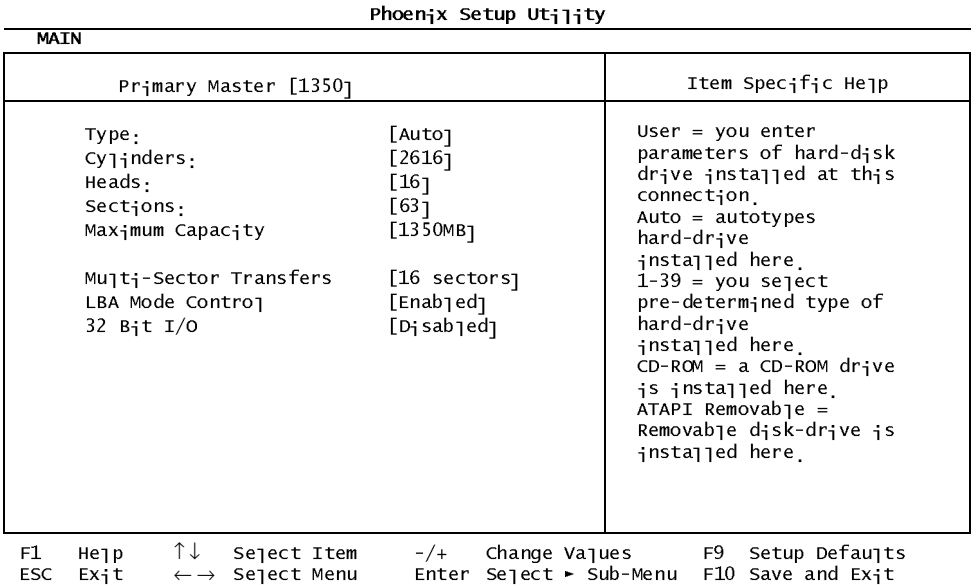
Use PgUp or Pgdn to select the floppy drive. The default is 1.44M, 3.5 inch.

Floppy Drive B

The VMIVME-7698 does not support a second floppy drive. The default is Disabled.

Primary Master/Slave

The VMIVME-7698 has the capability of utilizing one IDE hard disk drive on the Primary Master bus. The default setting is Auto. The Primary Slave is assigned to the CD-ROM (if installed).

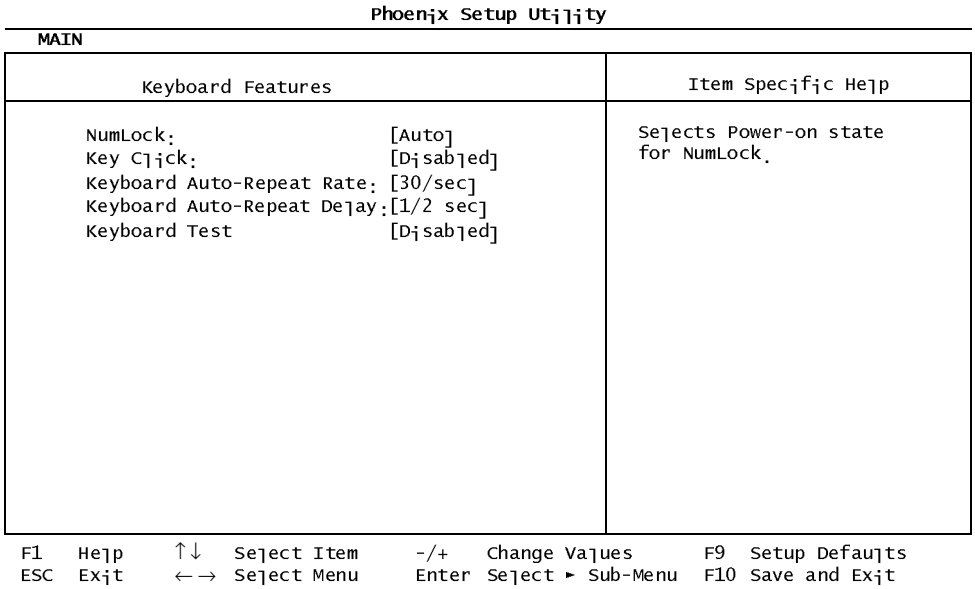


Secondary Master

The Secondary Master is the resident Flash Disk (if installed). The default setting is Auto.

Keyboard Features

The Keyboard Features allows the user to set several keyboard functions.



NumLock

The NumLock can be set to Auto, On, or Off to control the state of the NumLock key when the system boots. When set to Auto or On, the numeric keypad generates numbers instead of controlling the cursor operations. The default is On.

Key Click

This option enables or disables the Keyboard Auto-Repeat Rate and Delay settings. When disabled the values in the Typematic Rate and Delay are ignored. The default is Disabled.

Keyboard Auto-Repeat Rate (Chars/Sec)

If the Key Click is enabled this determines the rate a character is repeated when a key is held down. The options are: 30, 26.7, 21.8, 18.5, 13.3, 10, 6, or 2 characters per second. The default is 30.

Keyboard Auto-Repeat Delay (sec)

If the Key Click is enabled this determines the delay before a character starts repeating when a key is held down. The options are: 1/4, 1/2, 3/4, or 1 second. The default is 1/2.

Keyboard Test

When enabled this feature will test the keyboard during boot-up.

System Memory

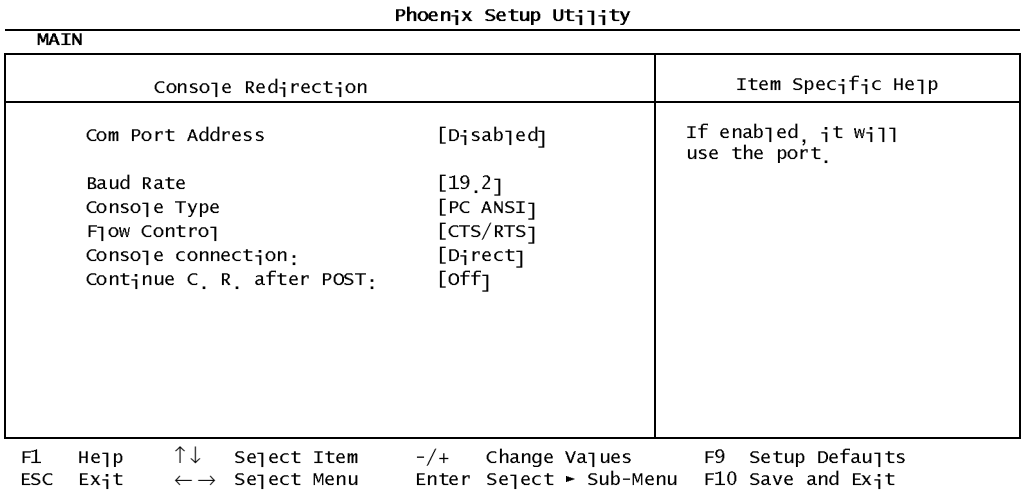
The System Memory field is for informational purposes only and cannot be modified by the user. This field displays the base memory installed in the system.

Extended Memory

The Extended Memory field is for informational purposes only and cannot be modified by the user. This field displays the total amount of memory installed in the system in Kbytes.

Console Redirection

Console Redirection allows for remote access and control of the PC functions to a remote terminal via the serial port. Selecting Console Redirection provides additional menus used to configure the console.



Com Port Address

If enabled, it will allow remote access through the serial port. The options are: Disabled, Motherboard Com A, and Motherboard Com B. The default is Disabled.

Baud Rate

Selects a baud rate for the serial port. The options are: 600, 1200, 2400, 4800, 9600, 19.2, 38.4, and 115.2. The default is 19.2.

Console Type

Selects the type of console to be used. The options are: PC ANSI or VT100. The default is PC ANSI.

Flow Control

Enables or disables Flow Control. The options are No Flow Control, XON / XOFF, or CTS / RTS. The default is CTS / RTS.

Console Connection

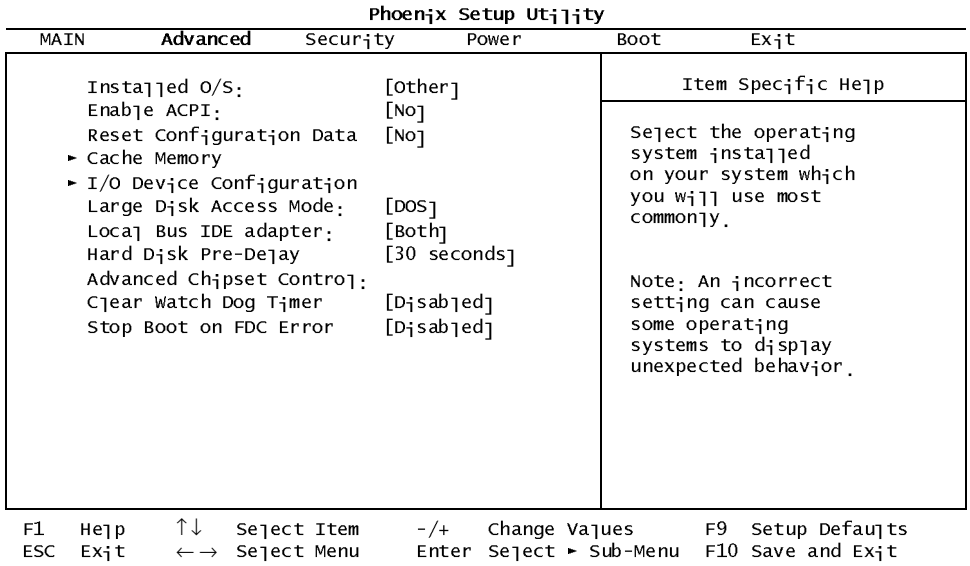
Indicates whether the console is connected directly to the system or if a modem is being used to connect. The options are: Direct or Via Modem. The default is Direct.

Continue C. R. After POST

This enables console redirection after the operating system has loaded. The options are OFF or ON. The default setting is OFF.

Advanced Menu

Selecting Advanced from the Main menu will display the screen shown below.



Installed O/S

Use this feature to select the operating system to use with your system.

Enable ACPI

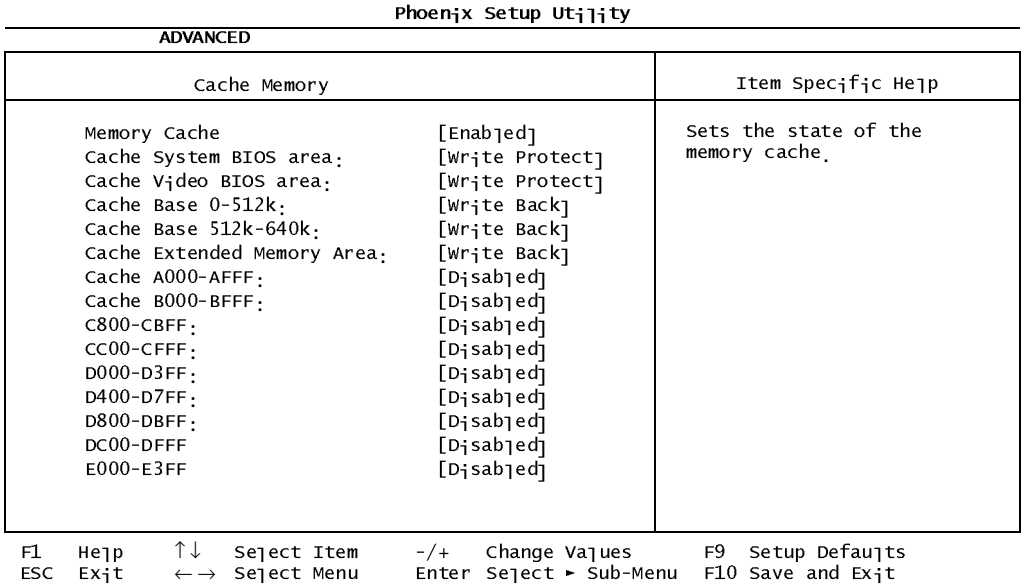
Enable or disable ACPI BIOS (Advance Configuration and Power Interface). Enable for ACPI OS only. The default is No.

Reset Configuration Data

Select Yes if you want to clear the extended system configuration data. The default is No.

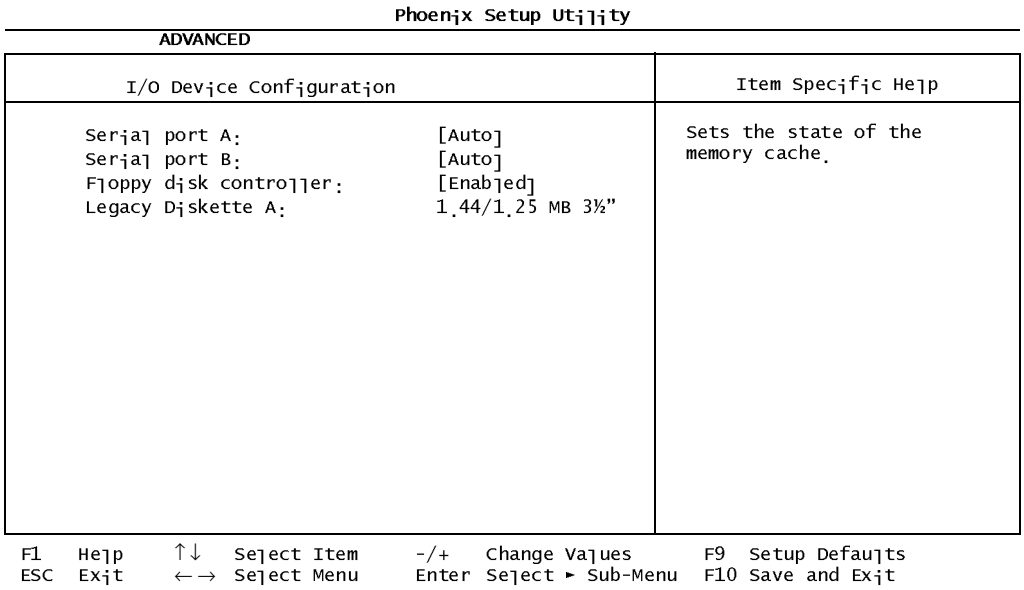
Cache Memory

Enabling the cache memory enhances the speed of the processor. When the CPU requests data, the system transfers the requested data from the main DRAM into the cache memory where it is stored until processed by the CPU. The default is Enabled.



I/O Device Configuration

Select this menu to configure your I/O devices, if required.



Large Disk Access Mode

The options for the Large Disk Access Mode are: UNIX Novell Netware or Other.

If you are installing new software and the drive fails, change this selection and try again. Different operating systems require different representations of drive geometries. The default is Other.

Local Bus IDE Adapter

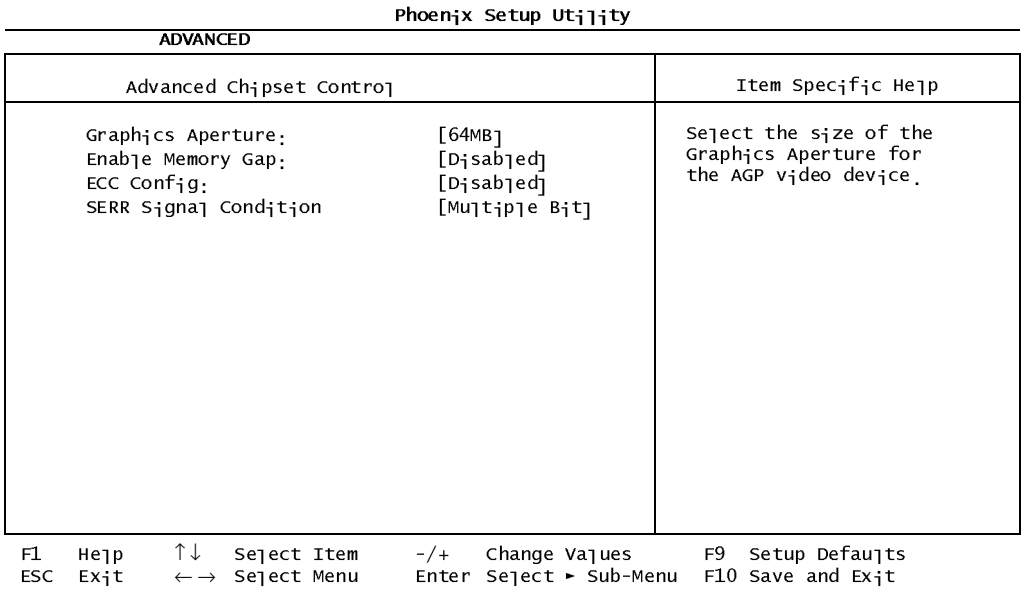
This enables or disables the intergrated local bus IDE adapter. The options are: Disabled, Primary, Secondary, or Both. The default is Both.

Hard Disk Pre-Delay

This feature adds a delay before the first access of a hard disk by the BIOS. Some hard disks will malfunction if accessed before they have initalized themselves. This delay ensures the hard disk has initialized after power-up, but prior to being accessed. The default is 30 seconds.

Advanced Chipset Control

Selecting Advanced Chipset Control opens the menu below. Use this menu to change the values in the chipset register for optimizing your system's performance.



Graphics Aperture

Select the size of the graphics aperture for the ACP video device. The options are 4MB, 8MB, 16MB, 64MB, 128MB, or 256MB. The default is 64MB.

Enable Memory Map

If enabled, turn system RAM off to free address space for use with an option card. Either a 128KB conventional memory gag, starting at 512KB, or a 1MB extended memory gap, starting at 15MB, will be created in the system RAM. The options are Disabled, Conventional, or Extended. The default is Disabled.

ECC Config

If all memory in the system supports ECC (x72) the ECC Config selects from No ECC (disabled), Checking Only (EC), Checking and Correction (ECC), or Checking, Correction with Scrubbing (ECC Scrub). The default is Disabled.

SERR Signal Configuration

Select ECC error conditions that SERR# will be asserted. The options are: None, Single Bit, Multiple Bit, or Both. The default is Both.

Clear Watchdog Timer Reset

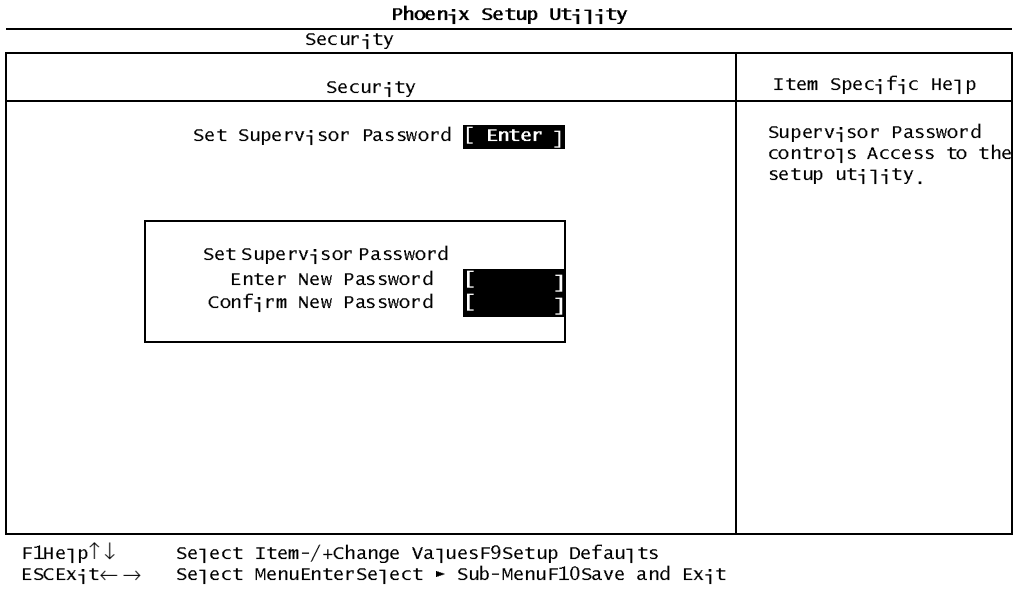
This enables or disables clearing of the Watchdog Timer at reset. The default is Disabled.

Stop Boot on FDC Error

When disabled this feature allows the boot process to continue without a floppy drive present. The default is Disabled.

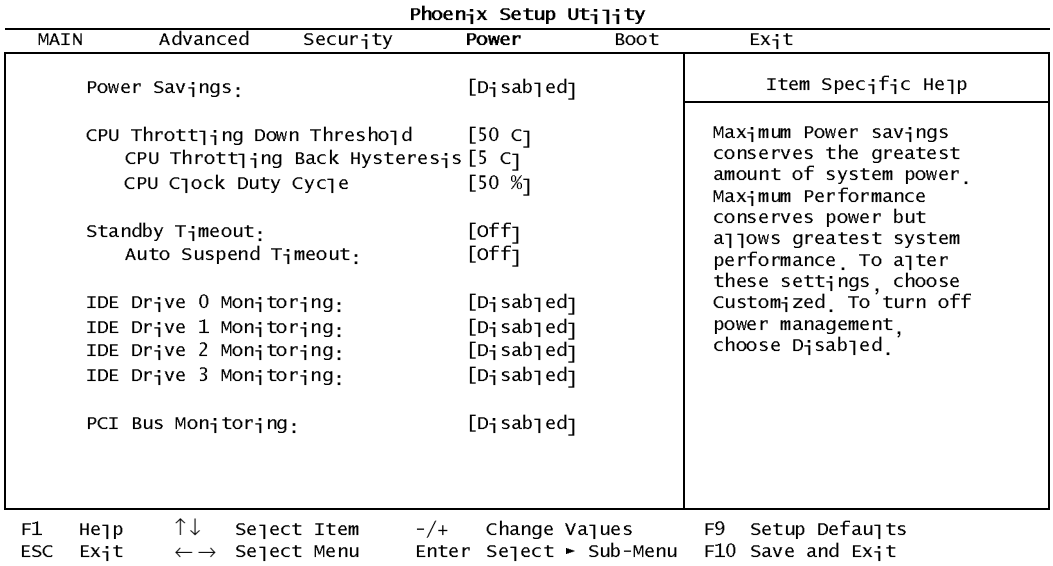
Security

Utilize this screen to set a user password.



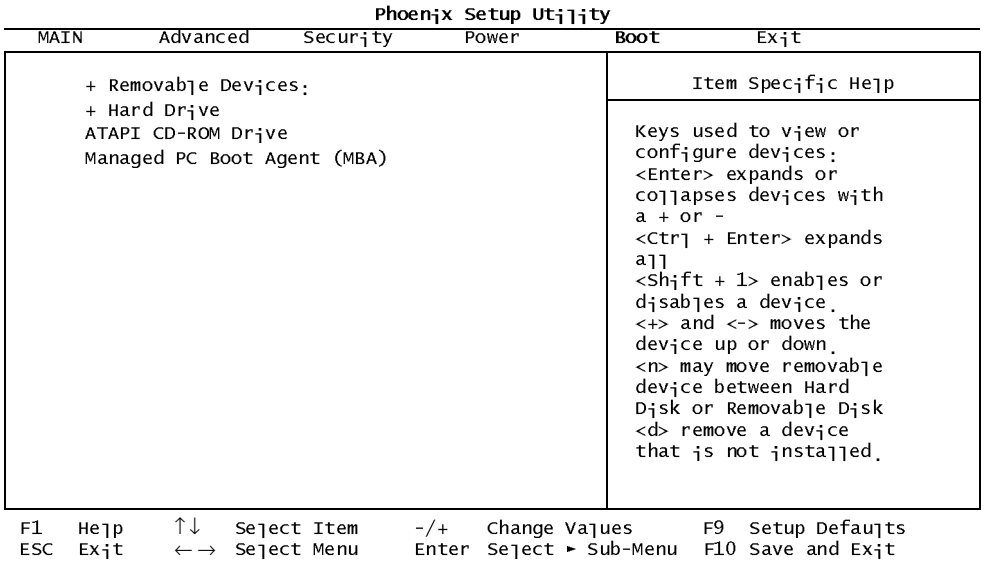
Power

This screen, selected from the Main screen, allows the user to configure power saving options on the VMIVME-7698.



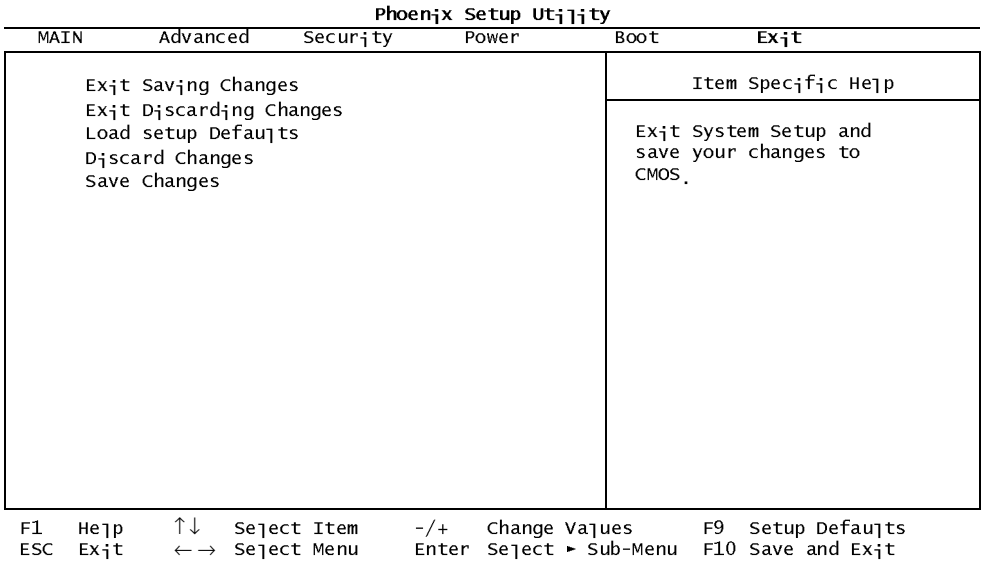
Boot Menu

The Boot priority is determined by the stack order, with the top having the highest priority and the bottom the least. The order can be modified by highlighting a device and, using the <+> or <-> keys, moving it to the desired order in the stack. A device can be boot disabled by highlighting the particular device and pressing <Shift 1>. <Enter> expands or collapses devices with a + or - next to them.



Exit Menu

The Exit menu allows the user to exit the BIOS program, while either saving or discarding any changes. This menu also allows the user to restore the BIOS defaults if desired.



Exit Saving Changes

Exit System Setup and save your changes to CMOS.

Exit Discarding Changes

Exit System Setup, discarding any changes to CMOS.

Load Setup Defaults

Load System defaults as defined at the factory.

Discard Changes

Discard any changes without exiting the setup program.

Save Changes

Save any changes made without exiting the Setup program.

Appendix

D

Device Configuration: I/O and Interrupt Control

Contents

Introduction

This appendix provides the user with the information needed to develop custom applications for the VMIVME-7698. The CPU board on the VMIVME-7698 is unique in that the BIOS can not be removed; it must be used in the initial boot cycle. A custom application, like a revised operating system for example, can only begin to operate after the BIOS has finished initializing the CPU. The VMIVME-7698 will allow the user to either maintain the current BIOS configuration or alter this configuration to be more user specific, but this alteration can only be accomplished after the initial BIOS boot cycle has completed.

BIOS Operations

When the VMIVME-7698 is powered on, control immediately jumps to the BIOS. The BIOS initiates a Power-on Self-Test (POST) program, which instructs the microprocessor to initialize system memory as well as the rest of the system. The BIOS establishes the configuration of all on-board devices by initializing their respective I/O and Memory addresses and interrupt request lines. The BIOS then builds an interrupt vector table in main memory, which is used for interrupt handling. The default interrupt vector table and the default address map are described in Chapter 3 of this manual. Finally, the BIOS jumps to the hard drive or floppy drive to execute the operating system's boot program. This is the point at which a custom operating system may take control of the board and proceed with a custom configuration and/or custom application. A user application may override the configuration set by the BIOS and reconfigure the system or it may accept what the BIOS initialized.

BIOS Control Overview

There are two areas on the VMIVME-7698 in which the user must be familiar in order to override the initial BIOS configuration. These include the device addresses and the device interrupts. This appendix reviews the details of these addresses and interrupts, and provides a reference list for the individual devices used on the board.

The VMIVME-7698 utilizes the high-performance Peripheral Component Interconnect (PCI) bus along with the Industrial Standard Architecture (ISA) bus. In general, the PCI bus is plug-and-play compatible. The components that are connected to the PCI bus are not always placed at a standard I/O or Memory address, nor are they connected to a standard interrupt request line as is the case with ISA bus devices. These PCI bus devices are re-established by the BIOS, meaning that these devices will not always be located at the same address or connected to the same interrupt request line every time the CPU is booted. This appendix lists the defaults that are found by powering up a specific VMIVME-7698.

Functional Overview

The block diagram included in Figure D-1 on page 109 illustrates the VMIVME-7698 emphasizing the I/O features, including the PCI-to-VMEbus bridge.

The circled number in the upper left corner of a function block references the appropriate data book necessary for the programming of the function block.

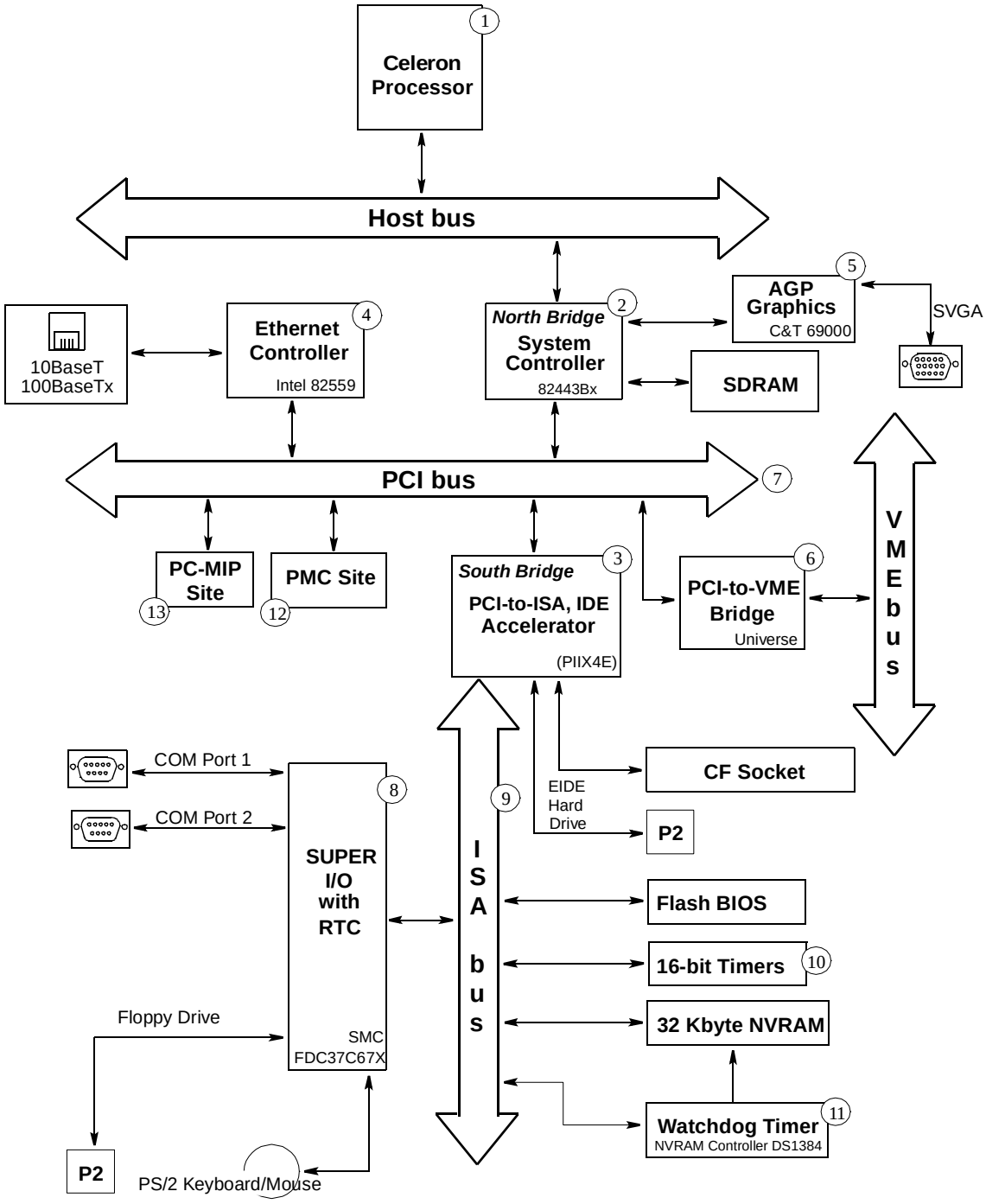


Figure D-1 VMIVME-7698 Block Diagram

Data Book References

1. *Celeron Processor Developer's Manual*
   Order Number 241428
   Intel Corporation
   2200 Mission College Blvd.
   P.O. Box 58119
   Santa Clara, CA 95052-8119
   (408) 765-8080
   www.intel.com

2. *Intel 440BX PCISet 82443BX Host Bridge/Controller*
   Intel Corporation
   2200 Mission College Boulevard
   P.O. Box 58119
   Santa Clara, CA 95052-8119

3. *Intel PIIX4E*
   82371EB PCI ISA IDE Xcellerator (PIIX4E)
   2200 Mission College Boulevard
   P.O. Box 58119
   Santa Clara, CA 95052-8119

4. *Intel 82559 10/100 Mb/s Ethernet LAN Controller*
   Intel
   www.intel.com

5. *Intel 69000 Technical Reference Manual*
   Intel Corporation
   2200 Mission College Blvd.
   P.O. Box 58119
   Santa Clara, CA 95052-8119
   (408) 765-8080
   www.intel.com

6. *VMIVME-7698 User Manuals*
   500-007698-000 Product Manual
   500-007698-001 VMIVME-7698, Tundra Universe II-Based VMEbus Interface Product Manual

7. *PCI Local Bus Specification, Rev. 2.1*
   PCI Special Interest Group
   P.O. Box 14070
   Portland, OR 97214
   (800) 433-5177 (U.S.)
   (503) 797-4207 (International)
   (503) 234-6762 (FAX)

8. *SMC FDC37C67X Enhanced Super I/O Controller*
SMC Component Products Division
300 Kennedy Drive
Hauppauge, NY 11788
(516) 435-6000
(516) 231-6004 (FAX)

9. *ISA & EISA, Theory and Operation*
Solari, Edward
Annabooks
15010 Avenue of Science, Suite 101
San Diego, CA 92128 USA
ISBN 0-929392 -15-9

10. *82C54 CHMOS Programmable Internal Timer*
Intel Corporation
2200 Mission College Boulevard
P.O. Box 58119
Santa Clara, CA 95052-8119

11. *DS 1384 Watchdog Timekeeping Controller*
Dallas Semiconductor
4461 South Beltwood Pwky
Dallas, TX 75244-3292

12. *IEEE P1386 PMC Specification*
The Institute of Electrical and Electronics Engineers, Inc.
345 East 47th Street
New York, NY 10017-2394
www.ieee.com

13. *Vita 29 Draft 0.92b PC-MIP Specification*
Vita Standards Organization
7825 East Gelding Drive
Suite 104
Scottsdale, AZ 85260
(602) 951-8866
www.vita.com

Device Address Definition

The standard PC / AT architecture defines two distinctive types of address spaces for the devices and peripherals on the CPU board. These spaces have typically been named Memory address space and I / O address space. The boundaries for these areas are limited to the number of address bus lines that are physically located on the CPU board. The VMIVME-7698 has 32 address bus lines located on the board, thereby defining the limit of the address space as 4 Gbyte. The standard PC / AT architecture defines Memory address space from zero to 4 Gbyte and the separate I / O address space from zero to 64 Kbyte.

ISA Devices

The ISA devices on the VMIVME-7698 are configured by the BIOS at boot-up and adhere to the standard PC / AT architecture. They are mapped in I / O address space within standard addresses and their interrupts are mapped to standard interrupt control registers. However, all of the ISA devices with the exception of the real-time clock, keyboard, and programmable timer are relocatable to almost anywhere within the standard 1 Kbyte of I / O address space. Table D-1 defines the spectrum of addresses available for reconfiguration of ISA devices.

As previously stated, in the standard PC / AT system, all I / O devices are mapped in I / O address space. On the VMIVME-7698, however, the Dynamic Random Access Memory (DRAM), Battery-Backed SRAM, and Watchdog Timer are addressed in Memory address space. The BIOS places DRAM at address zero and extends to the physical limit of the on-board DRAM.

Table D-1 ISA Device Mapping Configuration

Device	Memory Space	I/O Address Space	PIC Interrupt Options	Byte Address Boundary	Default
Floppy	N / A	[0x100 - 0xFF8]	IRQ1 - IRQ15	8	$3F0
Serial Port 1	N / A	[0x100 - 0xFF8]	IRQ1 - IRQ15	8	$3F8
Serial Port 2	N / A	[0x100 - 0xFF8]	IRQ1 - IRQ15	8	$2F8
Real-Time Clock	Nonrelocatable				$070
Keyboard	Nonrelocatable				$060
Auxiliary I / O	N / A	- Primary I / O [0x000 - 0xFFF] - Secondary I / O [0x000 - 0xFFF]	IRQ1, IRQ3-IRQ15	1 1	
Programmable Timer		Nonrelocatable 0x500	IRQ5	4	0x500
Battery-Backed SRAM	Nonrelocatable 0xD8018	N / A	N / A	(32K-0x18)	0xD8018

Table D-1 ISA Device Mapping Configuration (Continued)

Device	Memory Space	I/O Address Space	PIC Interrupt Options	Byte Address Boundary	Default
Board ID Register	Nonrelocatable 0xD8016	N / A	N / A	2	
VME BERR Address Modifier Register	Nonrelocatable 0xD8014	N / A	N / A	2	
VME BERR Address Register	Nonrelocatable 0xD8010	N / A		4	
System Command Register	Nonrelocatable 0xD800E	N / A		2	
Watchdog Timer	Nonrelocatable 0xD8000	N / A	N / A	0xD	0xD8000

PCI Devices

PCI devices are fully configured under I/O and/or Memory address space. Table D-2 describes the PCI bus devices that are on-board the VMIVME-7698 along with each device's configuration spectrum.

The PCI bus includes three physical address spaces. As with ISA bus, PCI bus supports Memory and I/O address space, but PCI bus includes an additional Configuration address space. This address space is defined to support PCI bus hardware configuration (refer to the PCI bus Specification for complete details on the configuration address space). PCI bus targets are required to implement Base Address registers in configuration address space to access internal registers or functions. The BIOS uses the Base Address register to determine how much space a device requires in a given address space and then assigns where in that space the device will reside. This functionality enables PCI devices to be located in either Memory or I/O address space.

Table D-2 PCI Device Mapping Configuration

Device	Memory Space	I/O Address Space	PIC Interrupt Options
SVGA	Anywhere	N / A	N / A
Universe II** PCI-to-VMEbus Bridge	Anywhere	N / A	PCI Defined
Ethernet	Anywhere	Anywhere	PCI Defined
Timer Interrupt Registers	N / A	Fixed	N / A
PC-MIP	Anywhere	Anywhere	PCI Defined
PMC Site	Anywhere	Anywhere	PCI Defined

** *Refer to the VMIVME-7698-001 User's Manual.*

Device Interrupt Definition

PC/AT Interrupt Definition

The interrupt hardware implementation on the VMIVME-7698 is standard for computers built around the PC / AT architecture. The PC / AT evolved from the IBM PC / XT architecture. In the IBM PC / XT systems, only eight Interrupt Request (IRQ) lines exist, numbered from IRQ0 to IRQ7. These interrupt lines were included originally on a 8259A Priority Interrupt Controller (PIC) chip.

The IBM PC / AT computer added eight more IRQx lines, numbered IRQ8 to IRQ15, by cascading a second slave 8259A PIC into the original master 8259A PIC. The interrupt line IRQ2 at the master PIC was committed as the cascade input from the slave PIC. This master / slave architecture, the standard PC / AT interrupt mapping, is illustrated in Figure D-2 on page 115 within the PCI-to-ISA Bridge PIIX4E 82371EB section of the diagram.

To maintain backward compatibility with PC / XT systems, IBM chose to use the new IRQ9 input on the slave PIC to operate as the old IRQ2 interrupt line on the PC / XT Expansion Bus. Thus, in AT systems, the IRQ9 interrupt line connects to the old IRQ2 pin on the AT Expansion Bus (or ISA bus).

The BIOS defines the PC / AT interrupt line to be used by each device. The BIOS writes to each of the two cascaded 8259A PIC chips an 8-bit vector, which maps each IRQx to its corresponding interrupt vector in memory.

ISA Device Interrupt Map

The VMIVME-7698 BIOS maps the IRQx lines to the appropriate device per the standard ISA architecture. Reference Figure D-2 on page 115. This initialization operation cannot be changed; however, a custom application could reroute the interrupt configuration after the BIOS has completed the initial configuration cycle.

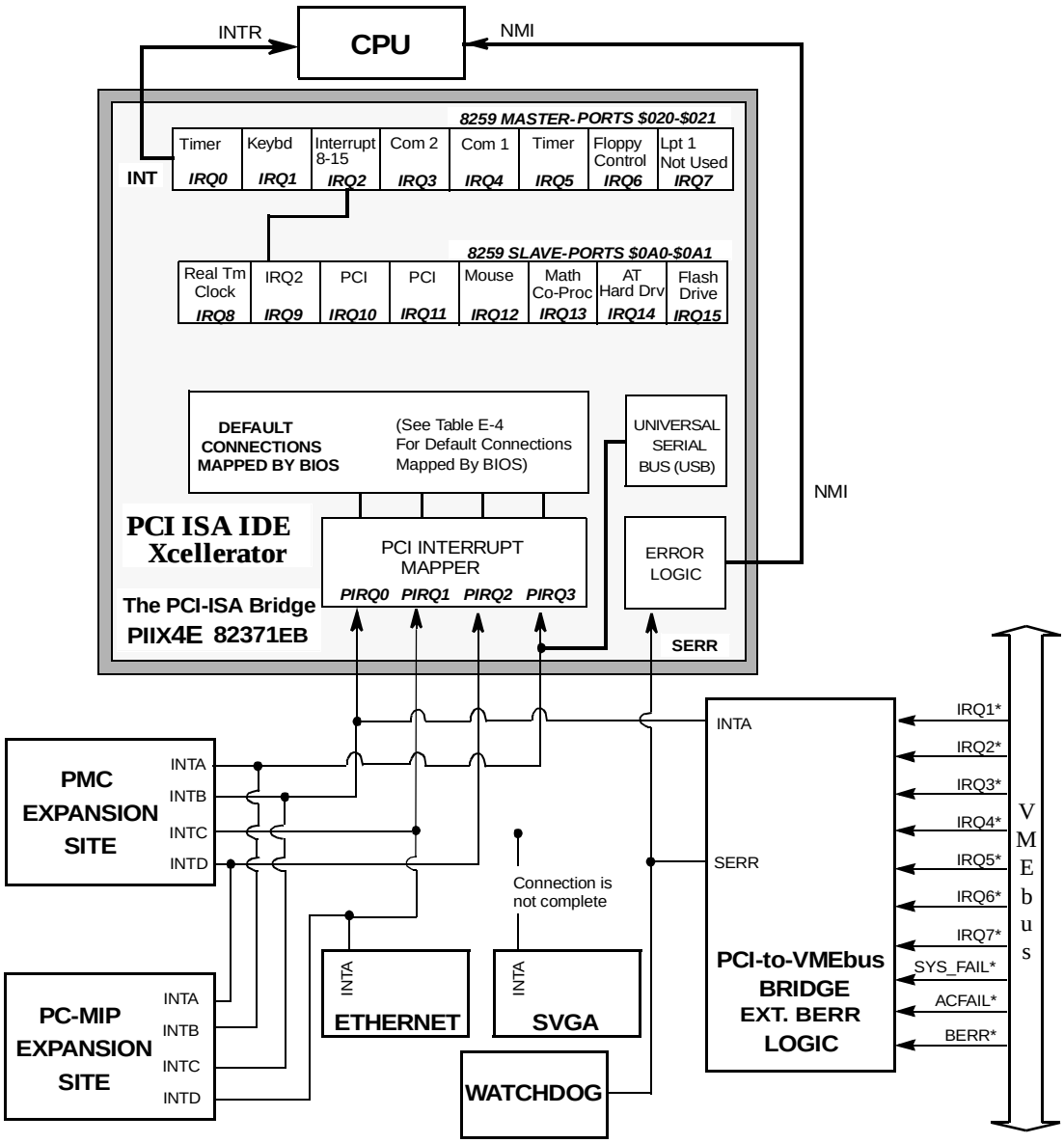


Figure D-2 BIOS Default Connections for the PC Interrupt Logic Controller

PCI Device Interrupt Map

The PCI bus-based external devices include the two PCI expansion sites, the PCI-to-VMEbus bridge, and the VGA reserved connection. The default BIOS maps these external devices to the PCI Interrupt Request (PIRQx) lines of the PIIX4E. This mapping is illustrated in Figure D-2 on page 115 and is defined in Table D-3.

The device PCI interrupt lines (INTA through INTD) that are present on each device *cannot* be modified.

Table D-3 Device PCI Interrupt Mapping by the BIOS

DEVICE	COMPONENT	VENDOR ID	DEVICE ID	CPU ADDRESS MAP ID SELECT	DEVICE PCI INTERRUPT	MOTHER-BOARD PCI INTERRUPT MAPPER	DATA BOOK REF. #	REVISION ID
PCI-to-VME Bridge Option: Tundra Universe II™	Universe CA91C142	0x10E3	0x0	AD19	INTA	PIRQ0	6	N / A
PCI Expansion Site	N / A	Board Specific	Board Specific	N / A	INTA	PIRQ3	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTB	PIRQ0	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTC	PIRQ1	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTD	PIRQ2	N / A	N / A
PC-MIP	N / A	Board Specific	Board Specific	N / A	INTA	PIRQ2	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTB	PIRQ3	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTC	PIRQ0	N / A	N / A
	N / A	Board Specific	Board Specific	N / A	INTD	PIRQ1	N / A	N / A
Power Management	PIIX4E 82371AB Function 03	0x8086	0x7113	AD18	N / A	N / A	3	N / A
PCI-to-ISA Bridge	PIIX4E 82371AB Function 00	0x8086	0x7110	AD18	N / A	N / A	3	N / A
SVGA Controller	Intel 69000	0x102C	0x0DC0	AD16	INTA**	PIRQ3**	5	N / A
Ethernet Controller	Intel 88559	0x8086	0x1229	AD20	INTA	PIRQ2	4	N / A
PCI IDE Controller	PIIX4E 82371AB Function 01	0x8086	0x7111	AD18	N / A	N / A	3	N / A
Universal Serial Bus (USB)***	PIIX4E 82371AB Function 02	0x8086	0x7112	AD18	INTD	PIRQ3	3	N / A
PCI Host Bridge	Intel 440BX	0x8086	0x7190	N / A	N / A	N / A	2	N / A
PCI to AGP Bridge	Intel 440BX	0x8086	0x7191	N / A	N / A	N / A	2	N / A

* To access these parts, use the revision number as the distinguishing factor.
** Not connected, for reference only
*** PIRQ4 interrupt is not enabled by the BIOS.

The motherboard accepts these PCI device interrupts through the PCI interrupt mapper function. The BIOS default maps the PCI Interrupt Request (PIRQx) external device lines to one of the available slave PIC Interrupt Request lines, IRQ (9, 10, 11, 12, 14, or 15).

Appendix

E

Sample C Software

Contents

Introduction

This appendix provides listings of a library of sample code that the programmer may utilize to build applications. These files are provided to the VMIVME-7698 user on disk 320-500037-006, Sample Application C Code for the VMIVME-7698, included in the distribution disk set.

Because of the wide variety of environments in which the VMIVME-7698 operates, the samples provided in this appendix are not necessarily intended to be verbatim boilerplates. Rather, they are intended to give the end user an example of the standard structure of the operating code.

Directory CPU

The code under the CPU directory sets up the universe chip with one PCI-TO-VME window and enables universe registers to be accessed from VME to allow mailbox access.

CPU.C

```
/*************************************************************************/
/*  FILE: CPU.C                                                          */
/*                                                                       */
/* Setup the universe chip with one PCI-TO-VME window and enable universe */
/* registers to be accessed from VME to allow mailbox access.            */
/*                                                                       */
/*************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include “flat.h”
#include “pci.h”
#include “universe.h”
#include “cpu.h”
#define   PCI_BASE16 0x10000000    /* PCI BASE for A16 */
#define   VME_16_RA   0x0000C000    /* VME BASE for A16 REG ACCESS */
#define   IRQ9      0x71        /* Int. No. for hardware int 9 */
#define   IRQA      0x72        /* Int. No. for hardware int A */
#define   IRQB      0x73        /* Int. No. for hardware int B */
#define   IRQC      0x74        /* Int. No. for hardware int C */
#define   IRQD      0x75        /* Int. No. for hardware int D */
#define   IRQE      0x76        /* Int. No. for hardware int E */
#define   IRQF      0x77        /* Int. No. for hardware int F */
```

```
/* function prototypes */
void far interrupt irq_rcvd( void );
void init_int( void );
void restore_orig_int( void );
void do_exit( int );
/* global variables */
unsigned char pic2_org;
unsigned long mb0_msg;
unsigned long mb1_msg;
unsigned long mb2_msg;
unsigned long mb3_msg;
unsigned long int_status;
void far interrupt (* old_vect)(void);
unsigned char int_line;
char user[80];
FPTR un_regs;
void main( void )
{
 unsigned char pci_devices;
 int test_int, to_cnt;
 unsigned long temp_dword;
 unsigned char bus, dev_func;
 printf("\n\n");
 /* try to locate the UNIVERSE device on the PCI bus */
 test_int = find_pci_device(UNIVERSE_DID, UNIVERSE_VID, 0,
            &bus, &dev_func);
 if(test_int == SUCCESSFUL)
 {
   test_int = read_configuration_area(READ_CONFIG_DWORD, bus, dev_func,
                        0x10, &temp_dword);
   if(test_int == SUCCESSFUL)
   {
```

```
      un_regs = (FPTR) temp_dword;
   }
     else
     {
      printf("Unable to read configuration area 0x10\n");
      exit(1);
   }
}
 else
 {
    printf("Unable to locate PCI device Tundra Universe\n");
    exit(1);
}
 test_int = read_configuration_area(READ_CONFIG_BYTE, bus, dev_func,
                                    0x3C, &temp_dword);
 if(test_int == SUCCESSFUL)
 {
    int_line = temp_dword & 0xFF;
}
 else
 {
    printf("Unable to read configuration area 0x3C\n");
    exit(1);
}
 /* setup protected mode */
 extend_seg();
 a20(1);
 mb0_msg = 0;
 mb1_msg = 0;
 mb2_msg = 0;
 mb3_msg = 0;
 init_int();
```

```
 /* 32K PCI slave window at 0x10000000 to VME A16 0x0000 user data */
 fw_long( un_regs + LSI0_BS_A, PCI_BASE16); /* pci base for A16 */
 fw_long(un_regs + LSI0_BD_A, (PCI_BASE16 + 0x8000)); /* 32K window */
 fw_long(un_regs + LSI0_TO_A, (0x00000000 - PCI_BASE16));
 fw_long(un_regs + LSI0_CTL_A, (LSI_CTL_EN | LSI_CTL_VDW_32 |
LSI_CTL_VAS_16));
 /* enable 4K VME to universe regs @ sht I/O 0xC000 to allow mailbox access */
 fw_long(un_regs + VRAI_BS_A, VME_16_RA);
 fw_long(un_regs + VRAI_CTL_A, (VRAI_CTL_EN | VRAI_CTL_AM_D |
                    VRAI_CTL_AM_US | VRAI_CTL_VAS_16));
 /* enable VME with master & slave set for big endian and time out 64 us */
 fw_word(CPUREGS, (VME_EN | MEC_BE | SEC_BE | BYPASS_EN | BTO_64 |
BTO_EN));
 /* place additional code here */
 do_exit(0);
} /* end main */
void do_exit(int xcode)
{
 /* disable all windows and ints */
 fw_long(un_regs + LSI0_CTL_A, 0);
 fw_long(un_regs + LSI1_CTL_A, 0);
 fw_long(un_regs + LSI2_CTL_A, 0);
 fw_long(un_regs + LSI3_CTL_A, 0);
 fw_long(un_regs + LSI4_CTL_A, 0);
 fw_long(un_regs + LSI5_CTL_A, 0);
 fw_long(un_regs + LSI6_CTL_A, 0);
 fw_long(un_regs + LSI7_CTL_A, 0);
 fw_long(un_regs + VSI0_CTL_A, 0);
 fw_long(un_regs + VSI1_CTL_A, 0);
 fw_long(un_regs + VSI2_CTL_A, 0);
 fw_long(un_regs + VSI3_CTL_A, 0);
 fw_long(un_regs + VSI4_CTL_A, 0);
 fw_long(un_regs + VSI5_CTL_A, 0);
```

```
 fw_long(un_regs + VSI6_CTL_A, 0);
 fw_long(un_regs + VSI7_CTL_A, 0);
 fw_long(un_regs + SLSI_A, 0);
 fw_long(un_regs + VRAI_CTL_A, 0);
 fw_long(un_regs + LINT_EN_A, 0);
 fw_word(CPUREGS, 0);
 restore_orig_int( );
 a20(0);
 exit(xcode);
} /* end do_exit */
/*****************************************************************/
/* init_int()                                                    */
/*                                                               */
/* purpose: Using the interrupt assigned, the original vector is */
/*          saved and the vector to the new ISR is installed. The */
/*          programmable-interrupt-controller (PIC) is enabled.  */
/*                                                               */
/*****************************************************************/
/* parameters: none                                              */
/*****************************************************************/
/* return value: none                                            */
/*****************************************************************/
void init_int(void)
{
 disable();
 /* Read 8259 slave Programmable Interrupt controller */
 pic2_org = inp(0xa1) & 0xFF; /* slave mask bits */
 switch(int_line)
 {
   case 0x9:
     old_vect = getvect(IRQ9); /* save vector for IRQ 09 */
     setvect(IRQ9, irq_rcvd);
```

```
  /* enable interrupt 9 */
  outp(0xa1, (pic2_org & 0xFD));
break;
case 0xa:
  old_vect = getvect(IRQA); /* save vector for IRQ 10 */
  setvect(IRQA, irq_rcvd);
  /* enable interrupt 10 */
  outp(0xa1, (pic2_org & 0xFB));
break;
case 0xb:
  old_vect = getvect(IRQB); /* save vector for IRQ 11 */
  setvect(IRQB, irq_rcvd);
  /* enable interrupt 11 */
  outp(0xa1, (pic2_org & 0xF7));
break;
case 0xc:
  old_vect = getvect(IRQC); /* save vector for IRQ 12 */
  setvect(IRQC, irq_rcvd);
  /* enable interrupt 12 */
  outp(0xa1, (pic2_org & 0xEF));
break;
case 0xd:
  old_vect = getvect(IRQD); /* save vector for IRQ 13 */
  setvect(IRQD, irq_rcvd);
  /* enable interrupt 13 */
  outp(0xa1, (pic2_org & 0xDF));
break;
case 0xe:
  old_vect = getvect(IRQE); /* save vector for IRQ 14 */
  setvect(IRQE, irq_rcvd);
  /* enable interrupt 14 */
  outp(0xa1, (pic2_org & 0xBF));
```

```
    break;
    case 0xf:
      old_vect = getvect(IRQF); /* save vector for IRQ 15 */
      setvect(IRQF, irq_rcvd);
      /* enable interrupt 15 */
      outp(0xa1, (pic2_org & 0x7F));
    break;
  } /* end switch */
  fw_long(un_regs + LINT_STAT_A, 0xFFF7FF); /* clear any previous status bits */
  fw_long(un_regs + LINT_MAP0_A, 0); /* map all VME ints to lint#0 INTA */
  fw_long(un_regs + LINT_MAP1_A, 0); /* map all ERR/STAT ints to lint#0 INTA */
  fw_long(un_regs + LINT_MAP2_A, 0); /* map all MB/LM ints to lint#0 INTA */
  /* enable mailbox 0 ints only */
  fw_long(un_regs + LINT_EN_A, LINT_EN_MBOX0);
  int_status = 0;
  enable();
} /* init_int */
/*********************************************************************/
/* restore_orig_int()                                                */
/*                                                                   */
/* purpose: Using the interrupt assigned, the original vector is     */
/*          restored and the programmable-interrupt-controller       */
/*          is restored to its original settings.                    */
/*                                                                   */
/* Prerequisite: The interrupt line to be used must have             */
/*              already been loaded in the global variable.          */
/*                                                                   */
/*********************************************************************/
/* parameters: none                                                  */
/*********************************************************************/
/* return value: none                                                */
/*********************************************************************/
```

```
void restore_orig_int(void)
{
  disable();
  outp(0xa1, pic2_org);
  switch( int_line )
  {
    case 0x9:
      setvect( IRQ9, old_vect );
    break;
    case 0xa:
      setvect( IRQA, old_vect );
    break;
    case 0xb:
      setvect( IRQB, old_vect );
    break;
    case 0xc:
      setvect( IRQC, old_vect );
    break;
    case 0xd:
      setvect( IRQD, old_vect );
    break;
    case 0xe:
      setvect( IRQE, old_vect );
    break;
    case 0xf:
      setvect( IRQF, old_vect );
    break;
  } /* end switch */
  fw_long( un_regs + LINT_EN_A, 0 );    /* disable all interrupts */
  enable();
} /* restore_orig_int */
```

```
/*****************************************************************/
/*  irq_rcvd()                                                   */
/*                                                               */
/*  purpose: Interrupt service routine. (INTA handler)           */
/*                                                               */
/*****************************************************************/
/*  parameters: none                                             */
/*****************************************************************/
/*  return value: none                                           */
/*****************************************************************/
void interrupt irq_rcvd( void )
{
 unsigned long lint_enable, tmp_status;
 disable();
 asm {
  .386P
  push  eax
  push  ebx
 }
 int_status = fr_long( un_regs + LINT_STAT_A ); /* read interrupt status */
 fw_long( un_regs + LINT_STAT_A, int_status );  /* clear status */
 /* check for mailbox interrupt */
 if( int_status & LINT_STAT_MBOX0 )
 {
  mb0_msg = fr_long( un_regs + MBOX0_A );
 }
 if( int_status & LINT_STAT_MBOX1 )
 {
  mb1_msg = fr_long( un_regs + MBOX1_A );
 }
 if( int_status & LINT_STAT_MBOX2 )
 {
```

```
  mb2_msg = fr_long( un_regs + MBOX2_A );
 }
 if( int_status & LINT_STAT_MBOX3 )
 {
  mb3_msg = fr_long( un_regs + MBOX3_A );
 }
 /* disable MB ints */
 lint_enable = fr_long( un_regs + LINT_EN_A );
 lint_enable &= ~(LINT_EN_MBOX3 | LINT_EN_MBOX2 |
          LINT_EN_MBOX1 | LINT_EN_MBOX0);
 fw_long( un_regs + LINT_EN_A, lint_enable );
 /* clear all mailboxes */
 fw_long( un_regs + MBOX0_A, 0 );
 fw_long( un_regs + MBOX1_A, 0 );
 fw_long( un_regs + MBOX2_A, 0 );
 fw_long( un_regs + MBOX3_A, 0 );
 tmp_status = fr_long( un_regs + LINT_STAT_A ); /* read interrupt status */
 fw_long( un_regs + LINT_STAT_A, tmp_status );  /* clear status */
 /* re-enable MB0 ints */
 lint_enable |= LINT_EN_MBOX0;
 fw_long( un_regs + LINT_EN_A, lint_enable );
 /* Non specific end of interrupt to master & slave PIC */
 outp(0x20, 0x20);  /* Master end of irq command */
 outp(0xa0, 0x20);  /* Slave end of irq command */
 asm {
  .386P
  pop   ebx
  pop   eax
 }
 enable();
}
```

** FILE: CPU.H

```
typedef unsigned char  Byte;
typedef unsigned short Word;
typedef unsigned long  Long;
/* universe Device ID and Vendor ID */
#define UNIVERSE_VID    0x10E3
#define UNIVERSE_DID    0x0000
/* CPU specific bits located at I/O 0x400 */
#define CPUREGS        0xD800E /* CPU  regs located at mem 0xD800E */
#define MEC_BE         0x0001  /* master endian conversion big endian */
#define MEC_LE         0x0000  /* master endian conversion little endian */
#define SEC_BE         0x0002  /* slave endian conversion big endian */
#define SEC_LE         0x0000  /* slave endian conversion little endian */
#define BERR_LATCH_EN   0x0004  /* buss error latch enable */
#define BTO_EN         0x0008  /* bus timeout timer enable */
#define BTO_16         0x0000  /* bus timeout 16 us */
#define BTO_64         0x0010  /* bus timeout 64 us */
#define BTO_256        0x0020  /* bus timeout 256 us */
#define BTO_1MS        0x0030  /* bus timeout 1 ms */
#define BERR_INT_EN     0x0040  /* bus error interrupt enable */
#define BERR_STAT_CLR   0x0080  /* buss error status/clear R/W1C */
#define WD_SYSFAIL      0x0100  /* watchdog to VME sysfail enable */
#define BYPASS_EN       0x0400  /* bypass enable - MEC/SEC must be LE */
#define VME_EN         0x0800  /* vme bus enable */
```

** FILE: FLAT.C

```
/*
** flat.c
**
** Access flat memory space (up to 4GB) in real mode.
**
*/
#include <stdio.h>
#include <dos.h>
#include "flat.h"
/*
** Keyboard controller defines
*/
#define RAMPORT    0x70
#define KB_PORT    0x64
#define PCNMIPORT  0xA0
#define INBA20     0x60
#define INBA20ON   0xDF
#define INBA20OFF  0xDD
/*
** macro to clear keyboard port
*/
#define kx() { while( inp( KB_PORT ) & 2 ); }
/*
** define local Global Descriptor Table (GDT) pointer structure
*/
static struct fword gdtptr;        /* fword ptr to gdt */
/*
** A20: Enable A20 line
**
** flag: enable  = 1
**       disable = 0
```

```
*/
void a20( int flag )
{
  kx();
  outp( KB_PORT, 0xD1 );
  kx();
  outp( INBA20, flag ? INBA20ON : INBA20OFF );
  kx();
  outp( KB_PORT, 0xFF );
  kx();
}
/*
**  convert a linear address to a far pointer
*/
void far * linear_to_seg( FPTR lin )
{
   void far *p;
   FP_SEG(p) = (unsigned int)( lin >> 4 );
   FP_OFF(p) = (unsigned int)( lin & 0xF );
   return p;
}
/*
**  Adjust the GS register's limit to 4GB
**
**  Note: interrupts are enabled by this call.
*/
void extend_seg( void )
{
  /*
  **  compute linear address and limit of GDT
  */
  gdtptr.linear_add = seg_to_linear(( void far * ) GDT );
```

```
  gdtptr.limit = 15;
  /*
  ** disable regular interrupts
  */
  disable();
  /*
  ** disable NMI
  */
  outp( RAMPORT, inp( RAMPORT ) | 0x80 );
  /*
  ** call protected mode code
  */
  protinit( &gdtptr );
  /*
  ** Turn interrupts back on
  */
  enable();
  /*
  ** Turn NMI back on
  */
  outp( RAMPORT, inp( RAMPORT ) & 0x7F );
}
void protinit( struct fword * address )
{
  asm {
    .386P
    push ds
    lds  bx,address
    lgdt FWORD ptr [bx]
    pop  ds
    mov  eax,cr0
    or   al,0x01
```

```
    mov   cr0,eax
    jmp   short nxt
   }
   nxt:
   asm {
    .386P
    mov   bx,8
    mov   gs,bx
    mov   es,bx
    and   al,0xfe
    mov   cr0,eax
   }
}
int fr_byte( FPTR adr )
{
 int d;
 asm {
  .386P
  xor   ax,ax      /* zero gs */
  mov   gs,ax
  mov   eax, adr
  mov   al,byte ptr gs:[eax]
  mov   d,ax
 }
 return d;
}
int fr_word( FPTR adr )
{
 int d;
 asm {
  .386P
  xor   ax,ax      /* zero gs */
```

```
    mov   gs,ax
    mov   eax, adr
    mov   ax,word ptr gs:[eax]
    mov   d,ax
  }
  return d;
}
long fr_long( FPTR adr )
{
  long  d;
  asm {
    .386P
    xor   ax,ax      /* zero gs */
    mov   gs,ax
    mov   eax,adr
    mov   eax,dword ptr gs:[eax]
    mov   d,eax
  }
  return d;
}
void fw_byte( FPTR a, int d )
{
  asm {
    .386P
    xor   ax,ax      /* zero gs */
    mov   gs,ax
    mov   eax,a
    mov   bx,d
    mov   byte ptr gs:[eax],bl
  }
}
void fw_word( FPTR a, int d )
```

```
{
 asm {
  .386P
  xor  ax,ax     /* zero gs */
  mov  gs,ax
  mov  eax,a
  mov  bx,d
  mov  word ptr gs:[eax],bx
 }
}
void fw_long( FPTR a, long d )
{
 asm {
  .386P
  xor  ax,ax     /* zero gs */
  mov  gs,ax
  mov  eax,a
  mov  ebx,d
  mov  dword ptr gs:[eax],ebx
}
}
/* flat move long */
void fml_string( FPTR d, FPTR s, long n )
{
 asm {
  .386P          /* have to use ES for string move */
  push es        /* save es */
  xor  ax,ax     /* zero gs */
  mov  gs,ax
  mov  es,ax
  mov  edi,d     /* This is the destination pointer */
  mov  esi,s     /* This is the source pointer */
```

```
    mov   ecx,n      /* This is the number of dwords */
    rep movs dword ptr es:[edi],dword ptr gs:[esi]
    pop es          /* give back es */
   }
}
/* flat move word */
void fmw_string( FPTR d, FPTR s, long n )
{
  asm {
    .386P           /* have to use ES for string move */
    push es         /* save es */
    xor   ax,ax     /* zero gs */
    mov   gs,ax
    mov   es,ax
    mov   edi,d     /* This is the destination pointer */
    mov   esi,s     /* This is the source pointer */
    mov   ecx,n     /* This is the number of words */
    rep movs word ptr es:[edi],word ptr gs:[esi]
    pop es          /* give back es */
   }
}
/* flat move byte */
void fmb_string( FPTR d, FPTR s, long n )
{
  asm {
    .386P           /* have to use ES for string move */
    push es         /* save es */
    xor   ax,ax     /* zero gs */
    mov   gs,ax
    mov   es,ax
    mov   edi,d     /* This is the destination pointer */
    mov   esi,s     /* This is the source pointer */
```

```
    mov   ecx,n       /* This is the number of bytes */
    rep movs byte ptr es:[edi],byte ptr gs:[esi]
    pop es            /* give back es */
  }
}
```

** FILE: FLAT.H

```
/*
**  flat.h
**
** Prototypes typedefs and macros for flat memory access
**
*/
typedef unsigned long FPTR;
/*
**  Global descriptor table
*/
struct _GDT {
  unsigned int limit;
  unsigned int base;
  unsigned int access;
  unsigned int hi_limit;
};
static struct _GDT GDT[2]= {
  {0,0,0,0},                    /* Null selector slot */
  {0xFFFF,0,0x9200,0x8F}        /* 4 Gig data segment */
};
/*
**  FWORD pointer to GDT
*/
struct fword {
  unsigned int limit;
  unsigned long linear_add;
};
/*
**  convert segmented address to linear address
*/
#define seg_to_linear(fp) (((FPTR) FP_SEG(fp)<<4)+FP_OFF(fp))
```

```
/*
** flat memory function prototypes
*/
void a20( int );
void far * linear_to_seg( FPTR );
void extend_seg( void );
void protinit( struct fword * );
int  fr_byte( FPTR );          /* flat read byte */
void fw_byte( FPTR, int );     /* flat write byte */
int  fr_word( FPTR );          /* flat read word */
void fw_word( FPTR, int );     /* flat write word */
long fr_long( FPTR );          /* flat read long */
void fw_long( FPTR, long );    /* flat write long */
void fml_string( FPTR, FPTR, long ); /* flat move long */
void fmw_string( FPTR, FPTR, long ); /* flat move word */
void fmb_string( FPTR, FPTR, long ); /* flat move byte */
```

** FILE: PCI.C

```
#include <dos.h>
#include <stddef.h>
#include "pci.h"
#define HIGH_BYTE(ax) (ax >> 8)
#define LOW_BYTE(ax) (ax & 0xff)
int find_pci_device(unsigned short device_id,
        unsigned short vendor_id,
        unsigned short index,
        unsigned char *bus_number,
        unsigned char *device_and_function)
{
  int ret_status;
  unsigned short ax, bx, flags;
  _CX = device_id;
  _DX = vendor_id;
  _SI = index;
  _AH = PCI_FUNCTION_ID;
  _AL = FIND_PCI_DEVICE;
  geninterrupt(0x1a);
  ax = _AX;
  bx = _BX;
  flags = _FLAGS;
  if ((flags & CARRY_FLAG) == 0)
  {
   ret_status = HIGH_BYTE(ax);
   if (ret_status == SUCCESSFUL)
   {
    if (bus_number != NULL) *bus_number = HIGH_BYTE(bx);
    if (device_and_function != NULL) *device_and_function = LOW_BYTE(bx);
   }
  }
```

```
  else
  {
   ret_status = NOT_SUCCESSFUL;
  }
  return(ret_status);
}
int read_configuration_area(unsigned char function,
                    unsigned char bus_number,
                    unsigned char device_and_function,
                    unsigned char register_number,
                    unsigned long *data)
{
  int ret_status;
  unsigned short ax, flags;
  unsigned long ecx;
  _BH = bus_number;
  _BL = device_and_function;
  _DI = register_number;
  _AH = PCI_FUNCTION_ID;
  _AL = function;
  geninterrupt(0x1a);
  ecx = _ECX;
  ax = _AX;
  flags = _FLAGS;
  if ((flags & CARRY_FLAG) == 0)
  {
   ret_status = HIGH_BYTE(ax);
   if (ret_status == SUCCESSFUL)
   {
    *data = ecx;
   }
  }
```

```
  else
  {
   ret_status = NOT_SUCCESSFUL;
  }
  return(ret_status);
}
int write_configuration_area(unsigned char function,
                unsigned char bus_number,
                unsigned char device_and_function,
                unsigned char register_number,
                unsigned long value)
{
  int ret_status;
  unsigned short ax, flags;
  _BH = bus_number;
  _BL = device_and_function;
  _ECX = value;
  _DI = register_number;
  _AH = PCI_FUNCTION_ID;
  _AL = function;
  geninterrupt(0x1a);
  ax = _AX;
  flags = _FLAGS;
  if ((flags & CARRY_FLAG) == 0)
  {
   ret_status = HIGH_BYTE(ax);
  }
  else
  {
   ret_status = NOT_SUCCESSFUL;
  }
  return(ret_status);
```

```
}
void outpd(unsigned short port, unsigned long value)
{
  _DX = port;
  _EAX = value;
  __emit__(0x66, 0xEF);
}
unsigned long inpd(unsigned short port)
{
  _DX = port;
  __emit__(0x66, 0xED);
  return(_EAX);
}
```

** FILE: PCI.H

```
#define TRUE 1
#define FALSE 0
#define CARRY_FLAG 0x01
/* PCI Functions */
#define PCI_FUNCTION_ID 0xB1
#define PCI_BIOS_PRESENT 0x01
#define FIND_PCI_DEVICE 0x02
#define FIND_PCI_CLASS_CODE 0x03
#define READ_CONFIG_BYTE 0x08
#define READ_CONFIG_WORD 0x09
#define READ_CONFIG_DWORD 0x0A
#define WRITE_CONFIG_BYTE 0x0B
#define WRITE_CONFIG_WORD 0x0C
#define WRITE_CONFIG_DWORD 0x0D
/* PCI Return codes */
#define SUCCESSFUL 0x00
#define NOT_SUCCESSFUL 0x01
/* Prototypes */
int find_pci_device(unsigned short device_id,
                    unsigned short vendor_id,
                    unsigned short index,
                    unsigned char *bus_number,
                    unsigned char *device_and_function);
int read_configuration_area(unsigned char function,
                        unsigned char bus_number,
                        unsigned char device_and_function,
                        unsigned char register_number,
                        unsigned long *data);
int write_configuration_area(unsigned char function,
                         unsigned char bus_number,
                         unsigned char device_and_function,
```

```
                    unsigned char register_number,
                    unsigned long value);
void outpd(unsigned short port, unsigned long value);
unsigned long inpd(unsigned short port);
```

** FILE: UNIVERSE.H

```
/*
**  file: universe.h
**
**
**  header file for the universe II chip register definitions
**
**
*/
typedef volatile struct universe_regs {
  unsigned long pci_id;       /* PCI device ID vendor ID              */
  unsigned long pci_csr;      /* PCI config control/status reg        */
  unsigned long pci_class;    /* PCI config class reg                 */
  unsigned long pci_misc0;    /* PCI config miscellaneous 0 reg       */
  unsigned long pci_bs0;      /* PCI config base address reg          */
  unsigned long pci_bs1;      /* PCI config base address reg          */
  unsigned long pci_u0[0x04]; /* unimplemented                        */
  unsigned long pci_r0[0x02]; /* reserved                             */
  unsigned long pci_u1;       /* unimplemented                        */
  unsigned long pci_r1[0x02]; /* reserved                             */
  unsigned long pci_misc1;    /* PCI config miscellaneous 1 reg       */
  unsigned long pci_u2[0x30]; /* unimplemented                        */
  unsigned long lsi0_ctl;     /* PCI slave image 0 control reg        */
  unsigned long lsi0_bs;      /* PCI slave image 0 base address reg   */
  unsigned long lsi0_bd;      /* PCI slave image 0 bound address reg  */
  unsigned long lsi0_to;      /* PCI slave image 0 translation offset reg */
  unsigned long ur0;          /* reserved                             */
  unsigned long lsi1_ctl;     /* PCI slave image 1 control reg        */
  unsigned long lsi1_bs;      /* PCI slave image 1 base address reg   */
  unsigned long lsi1_bd;      /* PCI slave image 1 bound address reg  */
  unsigned long lsi1_to;      /* PCI slave image 1 translation offset reg */
  unsigned long ur1;          /* reserved                             */
```

```
unsigned long lsi2_ctl;      /* PCI slave image 2 control reg          */
unsigned long lsi2_bs;       /* PCI slave image 2 base address reg     */
unsigned long lsi2_bd;       /* PCI slave image 2 bound address reg    */
unsigned long lsi2_to;       /* PCI slave image 2 translation offset reg */
unsigned long ur2;           /* reserved                               */
unsigned long lsi3_ctl;      /* PCI slave image 3 control reg          */
unsigned long lsi3_bs;       /* PCI slave image 3 base address reg     */
unsigned long lsi3_bd;       /* PCI slave image 3 bound address reg    */
unsigned long lsi3_to;       /* PCI slave image 3 translation offset reg */
unsigned long ur3[0x09];     /* reserved                               */
unsigned long scyc_ctl;      /* PCI special cycle control reg          */
unsigned long scyc_addr;     /* PCI special cycle PCI address reg      */
unsigned long scyc_en;       /* PCI special cycle swap/compare enable reg */
unsigned long scyc_cmp;      /* PCI special cycle compare data reg     */
unsigned long scyc_swp;      /* PCI special cycle swap data reg        */
unsigned long lmisc;         /* PCI miscellaneous reg                  */
unsigned long slsi;          /* PCI special PCI slave image            */
unsigned long l_cmderr;      /* PCI command error log reg              */
unsigned long laerr;         /* PCI address error log reg              */
unsigned long ur4[0x03];     /* reserved                               */
unsigned long lsi4_ctl;      /* PCI slave image 4 control reg          */
unsigned long lsi4_bs;       /* PCI slave image 4 base address reg     */
unsigned long lsi4_bd;       /* PCI slave image 4 bound address reg    */
unsigned long lsi4_to;       /* PCI slave image 4 translation offset reg */
unsigned long ur5;           /* reserved                               */
unsigned long lsi5_ctl;      /* PCI slave image 5 control reg          */
unsigned long lsi5_bs;       /* PCI slave image 5 base address reg     */
unsigned long lsi5_bd;       /* PCI slave image 5 bound address reg    */
unsigned long lsi5_to;       /* PCI slave image 5 translation offset reg */
unsigned long ur6;           /* reserved                               */
unsigned long lsi6_ctl;      /* PCI slave image 6 control reg          */
unsigned long lsi6_bs;       /* PCI slave image 6 base address reg     */
```

```
unsigned long lsi6_bd;      /* PCI slave image 6 bound address reg      */
unsigned long lsi6_to;      /* PCI slave image 6 translation offset reg */
unsigned long ur7;          /* reserved                                 */
unsigned long lsi7_ctl;     /* PCI slave image 7 control reg            */
unsigned long lsi7_bs;      /* PCI slave image 7 base address reg       */
unsigned long lsi7_bd;      /* PCI slave image 7 bound address reg      */
unsigned long lsi7_to;      /* PCI slave image 7 translation offset reg */
unsigned long ur8[0x05];    /* reserved                                 */
unsigned long dctl;         /* DMA transfer control reg                 */
unsigned long dtbc;         /* DMA transfer byte count reg              */
unsigned long dla;          /* DMA PCIbus address reg                   */
unsigned long ur9;          /* reserved                                 */
unsigned long dva;          /* DMA VMEbus address reg                   */
unsigned long urA;          /* reserved                                 */
unsigned long dcpp;         /* DMA command packet pointer               */
unsigned long urB;          /* reserved                                 */
unsigned long dgcs;         /* DMA general control and status reg       */
unsigned long d_llue;       /* DMA linked list update enable reg        */
unsigned long urC[0x36];    /* reserved                                 */
unsigned long lint_en;      /* PCI interrupt enable                     */
unsigned long lint_stat;    /* PCI interrupt status                     */
unsigned long lint_map0;    /* PCI interrupt map0                       */
unsigned long lint_map1;    /* PCI interrupt map1                       */
unsigned long vint_en;      /* VME interrupt enable                     */
unsigned long vint_stat;    /* VME interrupt status                     */
unsigned long vint_map0;    /* VME interrupt map0                       */
unsigned long vint_map1;    /* VME interrupt map1                       */
unsigned long statid;       /* VME interrupt status/ID out              */
unsigned long v1_statid;    /* VME interrupt status/ID in IRQ1          */
unsigned long v2_statid;    /* VME interrupt status/ID in IRQ2          */
unsigned long v3_statid;    /* VME interrupt status/ID in IRQ3          */
unsigned long v4_statid;    /* VME interrupt status/ID in IRQ4          */
```

```
unsigned long v5_statid;     /* VME interrupt status/ID in IRQ5        */
unsigned long v6_statid;     /* VME interrupt status/ID in IRQ6        */
unsigned long v7_statid;     /* VME interrupt status/ID in IRQ7        */
unsigned long lint_map2;     /* PCI interrupt map2                     */
unsigned long vint_map2;     /* VME interrupt map2                     */
unsigned long mbox0;         /* Mailbox 0                              */
unsigned long mbox1;         /* Mailbox 1                              */
unsigned long mbox2;         /* Mailbox 2                              */
unsigned long mbox3;         /* Mailbox 3                              */
unsigned long sema0;         /* Semaphore 0                            */
unsigned long sema1;         /* Semaphore 1                            */
unsigned long urD[0x28];     /* reserved                               */
unsigned long mast_ctl;      /* master control reg                     */
unsigned long misc_ctl;      /* miscellaneous control reg              */
unsigned long misc_stat;     /* miscellaneous status reg               */
unsigned long user_am;       /* user AM codes reg                      */
unsigned long urE[0x2bc];    /* reserved                               */
unsigned long vsi0_ctl;      /* VMEbus slave image 0 control reg       */
unsigned long vsi0_bs;       /* VMEbus slave image 0 base address reg  */
unsigned long vsi0_bd;       /* VMEbus slave image 0 bound address reg */
unsigned long vsi0_to;       /* VMEbus slave image 0 translation offset*/
unsigned long urF;           /* reserved                               */
unsigned long vsi1_ctl;      /* VMEbus slave image 1 control reg       */
unsigned long vsi1_bs;       /* VMEbus slave image 1 base address reg  */
unsigned long vsi1_bd;       /* VMEbus slave image 1 bound address reg */
unsigned long vsi1_to;       /* VMEbus slave image 1 translation offset*/
unsigned long urG;           /* reserved                               */
unsigned long vsi2_ctl;      /* VMEbus slave image 2 control reg       */
unsigned long vsi2_bs;       /* VMEbus slave image 2 base address reg  */
unsigned long vsi2_bd;       /* VMEbus slave image 2 bound address reg */
unsigned long vsi2_to;       /* VMEbus slave image 2 translation offset*/
unsigned long urH;           /* reserved                               */
```

```
unsigned long vsi3_ctl;      /* VMEbus slave image 3 control reg        */
unsigned long vsi3_bs;       /* VMEbus slave image 3 base address reg   */
unsigned long vsi3_bd;       /* VMEbus slave image 3 bound address reg  */
unsigned long vsi3_to;       /* VMEbus slave image 3 translation offset */
unsigned long urI[0x06];     /* reserved                                */
unsigned long lm_ctl;        /* Location Monitor Control                */
unsigned long lm_bs;         /* Location Monitor Base Address           */
unsigned long urJ;           /* reserved                                */
unsigned long vrai_ctl;      /* VMEbus register access image control reg */
unsigned long vrai_bs;       /* VMEbus register access image base address */
unsigned long urK[0x02];     /* reserved                                */
unsigned long vcsr_ctl;      /* VMEbus CSR control reg                  */
unsigned long vcsr_to;       /* VMEbus CSR translation reg              */
unsigned long v_amerr;       /* VMEbus AM code error log                */
unsigned long vaerr;         /* VMEbus address error log                */
unsigned long vsi4_ctl;      /* VMEbus slave image 4 control reg        */
unsigned long vsi4_bs;       /* VMEbus slave image 4 base address reg   */
unsigned long vsi4_bd;       /* VMEbus slave image 4 bound address reg  */
unsigned long vsi4_to;       /* VMEbus slave image 4 translation offset */
unsigned long urL;           /* reserved                                */
unsigned long vsi5_ctl;      /* VMEbus slave image 5 control reg        */
unsigned long vsi5_bs;       /* VMEbus slave image 5 base address reg   */
unsigned long vsi5_bd;       /* VMEbus slave image 5 bound address reg  */
unsigned long vsi5_to;       /* VMEbus slave image 5 translation offset */
unsigned long urM;           /* reserved                                */
unsigned long vsi6_ctl;      /* VMEbus slave image 6 control reg        */
unsigned long vsi6_bs;       /* VMEbus slave image 6 base address reg   */
unsigned long vsi6_bd;       /* VMEbus slave image 6 bound address reg  */
unsigned long vsi6_to;       /* VMEbus slave image 6 translation offset */
unsigned long urN;           /* reserved                                */
unsigned long vsi7_ctl;      /* VMEbus slave image 7 control reg        */
unsigned long vsi7_bs;       /* VMEbus slave image 7 base address reg   */
```

```
  unsigned long vsi7_bd;      /* VMEbus slave image 7 bound address reg   */
  unsigned long vsi7_to;      /* VMEbus slave image 7 translation offset  */
  unsigned long urP[0x05];    /* reserved                                 */
  unsigned long v_cr_csr;     /* VMEbus CR/CSR reserved                   */
  unsigned long vcsr_clr;     /* VMEbus CSR bit clear reg                 */
  unsigned long vcsr_set;     /* VMEbus CSR bit set reg                   */
  unsigned long vcsr_bs;      /* VMEbus CSR base address reg              */
} universe_regs_t;
#define   PCI_ID_A    0x000   /* PCI device ID vendor ID                  */
#define   PCI_CSR_A   0x004   /* PCI config control/status reg            */
#define   PCI_CLASS_A 0x008   /* PCI config class reg                     */
#define   PCI_MISC0_A 0x00C   /* PCI config miscellaneous 0 reg           */
#define   PCI_BS0_A   0x010   /* PCI config base address reg              */
#define   PCI_BS1_A   0x014   /* PCI config base address reg              */
#define   PCI_MISC1_A 0x03C   /* PCI config miscellaneous 1 reg           */
#define   LSI0_CTL_A  0x100   /* PCI slave image 0 control reg            */
#define   LSI0_BS_A   0x104   /* PCI slave image 0 base address reg       */
#define   LSI0_BD_A   0x108   /* PCI slave image 0 bound address reg      */
#define   LSI0_TO_A   0x10C   /* PCI slave image 0 translation offset reg */
#define   LSI1_CTL_A  0x114   /* PCI slave image 1 control reg            */
#define   LSI1_BS_A   0x118   /* PCI slave image 1 base address reg       */
#define   LSI1_BD_A   0x11C   /* PCI slave image 1 bound address reg      */
#define   LSI1_TO_A   0x120   /* PCI slave image 1 translation offset reg */
#define   LSI2_CTL_A  0x128   /* PCI slave image 2 control reg            */
#define   LSI2_BS_A   0x12C   /* PCI slave image 2 base address reg       */
#define   LSI2_BD_A   0x130   /* PCI slave image 2 bound address reg      */
#define   LSI2_TO_A   0x134   /* PCI slave image 2 translation offset reg */
#define   LSI3_CTL_A  0x13C   /* PCI slave image 3 control reg            */
#define   LSI3_BS_A   0x140   /* PCI slave image 3 base address reg       */
#define   LSI3_BD_A   0x144   /* PCI slave image 3 bound address reg      */
#define   LSI3_TO_A   0x148   /* PCI slave image 3 translation offset reg */
#define   SCYC_CTL_A  0x170   /* PCI special cycle control reg            */
```

```
#define   SCYC_ADDR_A  0x174   /* PCI special cycle PCI address reg       */
#define   SCYC_EN_A    0x178   /* PCI special cycle swap/compare enable reg */
#define   SCYC_CMP_A   0x17C   /* PCI special cycle compare data reg      */
#define   SCYC_SWP_A   0x180   /* PCI special cycle swap data reg         */
#define   LMISC_A      0x184   /* PCI miscellaneous reg                   */
#define   SLSI_A       0x188   /* PCI special PCI slave image             */
#define   L_CMDERR_A   0x18C   /* PCI command error log reg               */
#define   LAERR_A      0x190   /* PCI address error log reg               */
#define   LSI4_CTL_A   0x1A0   /* PCI slave image 4 control reg           */
#define   LSI4_BS_A    0x1A4   /* PCI slave image 4 base address reg      */
#define   LSI4_bd_A    0x1A8   /* PCI slave image 4 bound address reg     */
#define   LSI4_TO_A    0x1AC   /* PCI slave image 4 translation offset reg */
#define   LSI5_CTL_A   0x1B4   /* PCI slave image 5 control reg           */
#define   LSI5_BS_A    0x1B8   /* PCI slave image 5 base address reg      */
#define   LSI5_BD_A    0x1BC   /* PCI slave image 5 bound address reg     */
#define   LSI5_TO_A    0x1C0   /* PCI slave image 5 translation offset reg */
#define   LSI6_CTL_A   0x1C8   /* PCI slave image 6 control reg           */
#define   LSI6_BS_A    0x1CC   /* PCI slave image 6 base address reg      */
#define   LSI6_BD_A    0x1D0   /* PCI slave image 6 bound address reg     */
#define   LSI6_TO_A    0x1D4   /* PCI slave image 6 translation offset reg */
#define   LSI7_CTL_A   0x1DC   /* PCI slave image 7 control reg           */
#define   LSI7_BS_A    0x1E0   /* PCI slave image 7 base address reg      */
#define   LSI7_BD_A    0x1E4   /* PCI slave image 7 bound address reg     */
#define   LSI7_TO_A    0x1E8   /* PCI slave image 7 translation offset reg */
#define   DCTL_A       0x200   /* DMA transfer control reg                */
#define   DTBC_A       0x204   /* DMA transfer byte count reg             */
#define   DLA_A        0x208   /* DMA PCIbus address reg                  */
#define   DVA_A        0x210   /* DMA VMEbus address reg                  */
#define   DCPP_A       0x218   /* DMA command packet pointer              */
#define   DGCS_A       0x220   /* DMA general control and status reg      */
#define   D_LLUE_A     0x224   /* DMA linked list update enable reg       */
#define   LINT_EN_A    0x300   /* PCI interrupt enable                    */
```

```
#define  LINT_STAT_A  0x304   /* PCI interrupt status              */
#define  LINT_MAP0_A  0x308   /* PCI interrupt map0                */
#define  LINT_MAP1_A  0x30C   /* PCI interrupt map1                */
#define  VINT_EN_A    0x310   /* VME interrupt enable              */
#define  VINT_STAT_A  0x314   /* VME interrupt status              */
#define  VINT_MAP0_A  0x318   /* VME interrupt map0                */
#define  VINT_MAP1_A  0x31C   /* VME interrupt map1                */
#define  STATID_A     0x320   /* VME interrupt status/ID out       */
#define  V1_STATID_A  0x324   /* VME interrupt status/ID in IRQ1   */
#define  V2_STATID_A  0x328   /* VME interrupt status/ID in IRQ2   */
#define  V3_STATID_A  0x32C   /* VME interrupt status/ID in IRQ3   */
#define  V4_STATID_A  0x330   /* VME interrupt status/ID in IRQ4   */
#define  V5_STATID_A  0x334   /* VME interrupt status/ID in IRQ5   */
#define  V6_STATID_A  0x338   /* VME interrupt status/ID in IRQ6   */
#define  V7_STATID_A  0x33C   /* VME interrupt status/ID in IRQ7   */
#define  LINT_MAP2_A  0x340   /* PCI interrupt map2                */
#define  VINT_MAP2_A  0x344   /* VME interrupt map2                */
#define  MBOX0_A      0x348   /* Mailbox 0                         */
#define  MBOX1_A      0x34C   /* Mailbox 1                         */
#define  MBOX2_A      0x350   /* Mailbox 2                         */
#define  MBOX3_A      0x354   /* Mailbox 3                         */
#define  SEMA0_A      0x358   /* Semaphore 0                       */
#define  SEMA1_A      0x35C   /* Semaphore 1                       */
#define  MAST_CTL_A   0x400   /* master control reg                */
#define  MISC_CTL_A   0x404   /* miscellaneous control reg         */
#define  MISC_STAT_A  0x408   /* miscellaneous status reg          */
#define  USER_AM_A    0x40C   /* user AM codes reg                 */
#define  VSI0_CTL_A   0xF00   /* VMEbus slave image 0 control reg  */
#define  VSI0_BS_A    0xF04   /* VMEbus slave image 0 base address reg   */
#define  VSI0_BD_A    0xF08   /* VMEbus slave image 0 bound address reg  */
#define  VSI0_TO_A    0xF0C   /* VMEbus slave image 0 translation offset */
#define  VSI1_CTL_A   0xF14   /* VMEbus slave image 1 control reg  */
```

```
#define  VSI1_BS_A   0xF18  /* VMEbus slave image 1 base address reg    */
#define  VSI1_BD_A   0xF1C  /* VMEbus slave image 1 bound address reg   */
#define  VSI1_TO_A   0xF20  /* VMEbus slave image 1 translation offset  */
#define  VSI2_CTL_A  0xF28  /* VMEbus slave image 2 control reg         */
#define  VSI2_BS_A   0xF2C  /* VMEbus slave image 2 base address reg    */
#define  VSI2_BD_A   0xF30  /* VMEbus slave image 2 bound address reg   */
#define  VSI2_TO_A   0xF34  /* VMEbus slave image 2 translation offset  */
#define  VSI3_CTL_A  0xF3C  /* VMEbus slave image 3 control reg         */
#define  VSI3_BS_A   0xF40  /* VMEbus slave image 3 base address reg    */
#define  VSI3_BD_A   0xF44  /* VMEbus slave image 3 bound address reg   */
#define  VSI3_TO_A   0xF48  /* VMEbus slave image 3 translation offset  */
#define  LM_CTL_A    0xF64  /* Location Monitor Control                 */
#define  LM_BS_A     0xF68  /* Location Monitor Base Address            */
#define  VRAI_CTL_A  0xF70  /* VMEbus register access image control reg */
#define  VRAI_BS_A   0xF74  /* VMEbus register access image base address */
#define  VCSR_CTL_A  0xF80  /* VMEbus CSR control reg                   */
#define  VCSR_TO_A   0xF84  /* VMEbus CSR translation reg               */
#define  V_AMERR_A   0xF88  /* VMEbus AM code error log                 */
#define  VAERR_A     0xF8C  /* VMEbus address error log                 */
#define  VSI4_CTL_A  0xF90  /* VMEbus slave image 4 control reg         */
#define  VSI4_BS_A   0xF94  /* VMEbus slave image 4 base address reg    */
#define  VSI4_BD_A   0xF98  /* VMEbus slave image 4 bound address reg   */
#define  VSI4_TO_A   0xF9C  /* VMEbus slave image 4 translation offset  */
#define  VSI5_CTL_A  0xFA4  /* VMEbus slave image 5 control reg         */
#define  VSI5_BS_A   0xFA8  /* VMEbus slave image 5 base address reg    */
#define  VSI5_BD_A   0xFAC  /* VMEbus slave image 5 bound address reg   */
#define  VSI5_TO_A   0xFB0  /* VMEbus slave image 5 translation offset  */
#define  VSI6_CTL_A  0xFB8  /* VMEbus slave image 6 control reg         */
#define  VSI6_BS_A   0xFBC  /* VMEbus slave image 6 base address reg    */
#define  VSI6_BD_A   0xFC0  /* VMEbus slave image 6 bound address reg   */
#define  VSI6_TO_A   0xFC4  /* VMEbus slave image 6 translation offset  */
#define  VSI7_CTL_A  0xFCC  /* VMEbus slave image 7 control reg         */
```

```
#define   VSI7_BS_A   0xFD0   /* VMEbus slave image 7 base address reg     */
#define   VSI7_BD_A   0xFD4   /* VMEbus slave image 7 bound address reg    */
#define   VSI7_TO_A   0xFD8   /* VMEbus slave image 7 translation offset   */
#define   V_CR_CSR_A  0xFF0   /* VMEbus CR/CSR reserved                    */
#define   VCSR_CLR_A  0xFF4   /* VMEbus CSR bit clear reg                  */
#define   VCSR_SET_A  0xFF8   /* VMEbus CSR bit set reg                    */
#define   VCSR_BS_A   0xFFC   /* VMEbus CSR base address reg               */
/* PCI and VME slave window structure */
typedef struct slave_window {
 unsigned long win_ctl;
 unsigned long win_bs;
 unsigned long win_bd;
 unsigned long win_to;
} swin_config_t;
/* DMA command packet structure for dmas using linked lists */
typedef struct dma_command_pkt {
 unsigned long dma_dctl;     /* DMA transfer control reg              */
 unsigned long dma_dtbc;     /* DMA transfer byte count reg           */
 unsigned long dma_dla;      /* DMA PCI bus address reg               */
 unsigned long rsvd1;        /* RESERVED                              */
 unsigned long dma_dva;      /* DMA VMEbus address reg                */
 unsigned long rsvd2;        /* RESERVED                              */
 unsigned long dma_dcpp;     /* DMA command packet pointer reg        */
 unsigned long rsvd3;        /* RESERVED                              */
} dma_cmd_pkt_t;
/* pci_id - PCI device ID and vendor ID */
#define   PCI_DID_VID   0x000010E3  /* R PCI device ID vendor ID       */
/* pci_csr - PCI configuration space control and status register */
#define   PCI_CSR_D_PE  0x80000000  /* R/WC detected parity error      */
#define   PCI_CSR_S_SERR 0x40000000  /* R/WC signalled SERR*           */
#define   PCI_CSR_R_MA  0x20000000  /* R/WC received master abort      */
#define   PCI_CSR_R_TA  0x10000000  /* R/WC received target abort      */
```

```
#define    PCI_CSR_S_TA    0x08000000  /* R/WC signalled target abort      */
#define    PCI_CSR_DEVSEL  0x02000000  /* R device select timing - medium  */
#define    PCI_CSR_DP_D    0x01000000  /* RC data parity detected          */
#define    PCI_CSR_TFBBC   0x00800000  /* R target fast back-to-back capable */
#define    PCI_CSR_MFBBC   0x00000000  /* R master fast back-to-back capable */
#define    PCI_CSR_SERR_EN 0x00000100  /* R/W SERR* enable                 */
#define    PCI_CSR_WAIT    0x00000080  /* R wait cycle control             */
#define    PCI_CSR_PERESP  0x00000040  /* R/W parity error response        */
#define    PCI_CSR_VGAPS   0x00000020  /* R VGA palette snoop              */
#define    PCI_CSR_MWI_EN  0x00000010  /* R mem write and invalidate enable */
#define    PCI_CSR_SC      0x00000008  /* R special cycles                 */
#define    PCI_CSR_BM      0x00000004  /* R/W master enable                */
#define    PCI_CSR_MS      0x00000002  /* R/W target memory enable         */
#define    PCI_CSR_IOS     0x00000001  /* R/W target I/O enable            */
/* pci_class - PCI configuration class register */
#define    PCI_CLASS_BASE  0x06000000  /* R base class code                */
#define    PCI_CLASS_SUB   0x00800000  /* R sub class code                 */
#define    PCI_CLASS_PROG  0x00000000  /* R programming interface          */
#define    PCI_CLASS_RID   0x00000000  /* R revision ID                    */
/* pci_misc0 - PCI configuration miscellanious 0 register */
#define    PCI_MISC0_BISTC  0x80000000  /* R BIST capable N/A              */
#define    PCI_MISC0_SBIST  0x40000000  /* R start BIST  N/A               */
#define    PCI_MISC0_CCODE  0x0F000000  /* R completion code MASK          */
#define    PCI_MISC0_MFUNCT 0x00800000  /* R multifunction device          */
#define    PCI_MISC0_LAYOUT 0x007F0000  /* R configuration space layout MASK
*/
#define    PCI_MISC0_LTIMER 0x0000F800  /* R/W latency timer MASK          */
/* pci_bs - PCI configuration base address register */
#define    PCI_BS_BS       0xFFFF0000  /* R PCI base address MASK         */
#define    PCI_BS_SPACE_M  0x00000000  /* R PCI address space memory      */
#define    PCI_BS_SPACE_IO 0x00000001  /* R PCI address space I/O         */
/* pci_misc1 - PCI configuration miscellaneous 1 register */
```

```
#define    PCI_MISC1_MAX_LAT  0x00000000  /* R maximum latency: none       */
#define    PCI_MISC1_MAX_GNT  0x00030000  /* R minimum grant: 250 ns       */
#define    PCI_MISC1_INT_PIN  0x00000100  /* R interrupt pin               */
#define    PCI_MISC1_INT_LINE 0x000000FF  /* R/W interrupt line MASK       */
/* LSI[X]_ctl - slave image control registers ( lsi0 - lsi7 ) */
#define    LSI_CTL_EN     0x80000000  /* R/W image enable                  */
#define    LSI_CTL_PWEN   0x40000000  /* R/W posted write enable           */
#define    LSI_CTL_VDW_08 0x00000000  /* R/W VMEbus maximum data width
D08  */
#define    LSI_CTL_VDW_16 0x00400000  /* R/W VMEbus maximum data width
D16  */
#define    LSI_CTL_VDW_32 0x00800000  /* R/W VMEbus maximum data width
D32  */
#define    LSI_CTL_VDW_64 0x00C00000  /* R/W VMEbus maximum data width
D64  */
#define    LSI_CTL_VAS_16 0x00000000  /* R/W VMEbus address space A16      */
#define    LSI_CTL_VAS_24 0x00010000  /* R/W VMEbus address space A24      */
#define    LSI_CTL_VAS_32 0x00020000  /* R/W VMEbus address space A32      */
#define    LSI_CTL_VAS_R1 0x00030000  /* R/W VMEbus address space RSVD1    */
#define    LSI_CTL_VAS_R2 0x00040000  /* R/W VMEbus address space RSVD2    */
#define    LSI_CTL_VAS_CR 0x00050000  /* R/W VMEbus address space CR/CSR
*/
#define    LSI_CTL_VAS_U1 0x00060000  /* R/W VMEbus address space USER1    */
#define    LSI_CTL_VAS_U2 0x00070000  /* R/W VMEbus address space USER2    */
#define    LSI_CTL_PGM_D  0x00000000  /* R/W VMEbus data AM code           */
#define    LSI_CTL_PGM_P  0x00004000  /* R/W VMEbus program AM code        */
#define    LSI_CTL_SUPER  0x00001000  /* R/W VMEbus supervisory AM code    */
#define    LSI_CTL_VCT_S  0x00000000  /* R/W VMEbus single cycles only     */
#define    LSI_CTL_VCT_SB 0x00000100  /* R/W VMEbus single cycles and block */
#define    LSI_CTL_LAS_M  0x00000000  /* R/W PCIbus memory space           */
#define    LSI_CTL_LAS_IO 0x00000001  /* R/W PCIbus I/O space              */
#define    LSI_CTL_LAS_C  0x00000002  /* R/W PCIbus type 1 config space    */
#define    LSI_CTL_LAS_R  0x00000003  /* R/W PCIbus reserved               */
/* lsi[X]_bs - slave image 0/1/2/3/4/5/6/7 base address register 0x0000?XXX */
```

```
#define    LSI0_BS    0xFFFFF000  /* R/W PCI slave image 0 base address MASK */
#define    LSI1_BS    0xFFFF0000  /* R/W PCI slave image 1 base address MASK */
#define    LSI2_BS    0xFFFF0000  /* R/W PCI slave image 2 base address MASK */
#define    LSI3_BS    0xFFFF0000  /* R/W PCI slave image 3 base address MASK */
#define    LSI4_BS    0xFFFFF000  /* R/W PCI slave image 4 base address MASK */
#define    LSI5_BS    0xFFFF0000  /* R/W PCI slave image 5 base address MASK */
#define    LSI6_BS    0xFFFF0000  /* R/W PCI slave image 6 base address MASK */
#define    LSI7_BS    0xFFFF0000  /* R/W PCI slave image 7 base address MASK */
/* lsi[X]_bd - slave image 0/1/2/3/4/5/6/7 bound address register 0x0000?XXX */
#define    LSI0_BD    0xFFFFF000  /* R/W PCI slave image 0 bound addr MASK  */
#define    LSI1_BD    0xFFFF0000  /* R/W PCI slave image 1 bound addr MASK  */
#define    LSI2_BD    0xFFFF0000  /* R/W PCI slave image 2 bound addr MASK  */
#define    LSI3_BD    0xFFFF0000  /* R/W PCI slave image 3 bound addr MASK  */
#define    LSI4_BD    0xFFFFF000  /* R/W PCI slave image 4 bound addr MASK  */
#define    LSI5_BD    0xFFFF0000  /* R/W PCI slave image 5 bound addr MASK  */
#define    LSI6_BD    0xFFFF0000  /* R/W PCI slave image 6 bound addr MASK  */
#define    LSI7_BD    0xFFFF0000  /* R/W PCI slave image 7 bound addr MASK  */
/* lsi[X]_to - slave image 0/1/2/3/4/5/6/7 translation offset reg 0x0000?XXX */
#define    LSI0_TO    0xFFFFF000  /* R/W PCI slave image 0 xfer offset MASK */
#define    LSI1_TO    0xFFFF0000  /* R/W PCI slave image 1 xfer offset MASK */
#define    LSI2_TO    0xFFFF0000  /* R/W PCI slave image 2 xfer offset MASK */
#define    LSI3_TO    0xFFFF0000  /* R/W PCI slave image 3 xfer offset MASK */
#define    LSI4_TO    0xFFFFF000  /* R/W PCI slave image 4 xfer offset MASK */
#define    LSI5_TO    0xFFFF0000  /* R/W PCI slave image 5 xfer offset MASK */
#define    LSI6_TO    0xFFFF0000  /* R/W PCI slave image 6 xfer offset MASK */
#define    LSI7_TO    0xFFFF0000  /* R/W PCI slave image 7 xfer offset MASK */
/* scyc_ctl - spceial cycle control register */
#define    SCYC_CTL_MEM   0x00000000  /* R/W PCI bus Memory space        */
#define    SCYC_CTL_IO    0x00000004  /* R/W PCI bus I/O space           */
#define    SCYC_CTL_DIS   0x00000000  /* R/W special cycle disabled      */
#define    SCYC_CTL_RMW   0x00000001  /* R/W read-modify-write           */
#define    SCYC_CTL_ADOH  0x00000002  /* R/W address only                */
```

```
#define    SCYC_CTL_RSVD  0x00000003  /* R/W reserved                */
/* scyc_addr - special cycle PCI bus address register 0x0000000X */
#define    SCYC_ADDR   0xFFFFFFFC  /* R/W spceial cycle PCIbus add reg MASK
*/
/* scyc_en - special cycle swap/compare enable reister 0x00000000 */
#define    SCYC_EN     0xFFFFFFFF  /* R/W spceial cycle swap/compare en MASK
*/
/* scyc_cmp - special cycle compare data register 0x00000000 */
#define    SCYC_CMP    0xFFFFFFFF  /* R/W spceial cycle compare data MASK   */
/* scyc_swp - special cycle swap data register 0x00000000 */
#define    SCYC_SWP    0xFFFFFFFF  /* R/W spceial cycle swap data MASK      */
/* lmisc - PCI miscellaneous register */
#define    LMISC_CRT_D 0x00000000  /* R/W coupled request timer disabled    */
#define    LMISC_CRT_1 0x10000000  /* R/W coupled request timer 128 us      */
#define    LMISC_CRT_2 0x20000000  /* R/W coupled request timer 256 us      */
#define    LMISC_CRT_3 0x30000000  /* R/W coupled request timer 512 us      */
#define    LMISC_CRT_4 0x40000000  /* R/W coupled request timer 1024 us     */
#define    LMISC_CRT_5 0x50000000  /* R/W coupled request timer 2048 us     */
#define    LMISC_CRT_6 0x60000000  /* R/W coupled request timer 4096 us     */
#define    LMISC_CWT_D 0x00000000  /* R/W coupled window timer disabled     */
#define    LMISC_CWT_1 0x01000000  /* R/W coupled window timer 128 us       */
#define    LMISC_CWT_2 0x02000000  /* R/W coupled window timer 256 us       */
#define    LMISC_CWT_3 0x03000000  /* R/W coupled window timer 512 us       */
#define    LMISC_CWT_4 0x04000000  /* R/W coupled window timer 1024 us      */
#define    LMISC_CWT_5 0x05000000  /* R/W coupled window timer 2048 us      */
#define    LMISC_CWT_6 0x06000000  /* R/W coupled window timer 4096 us      */
/* slsi - special PCI slave image */
#define    SLSI_EN     0x80000000  /* R/W image enable                 */
#define    SLSI_PWEN   0x40000000  /* R/W posted write enable          */
#define    SLSI_VDW    0x00F00000  /* R/W VME max data width MASK      */
#define    SLSI_PGM    0x0000F000  /* R/W VME program/data AM code MASK
*/
```

```
#define    SLSI_SUPER  0x00000F00  /* R/W VME supervisor/user AM code MASK
*/
#define    SLSI_BS     0x000000FC  /* R/W base address MASK              */
#define    SLSI_LAS_M  0x00000000  /* R/W PCIbus memory space            */
#define    SLSI_LAS_IO 0x00000001  /* R/W PCIbus I/O space               */
#define    SLSI_LAS_C  0x00000002  /* R/W PCIbus type 1 configuration space */
#define    SLSI_LAS_R  0x00000003  /* R/W PCIbus reserved                */
/* l_cmderr - PCI command error log register */
#define    L_CMDERR_CMDERR 0xF0000000  /* R PCI command error log MASK
*/
#define    L_CMDERR_M_ERR  0x08000000  /* R multiple error occurred      */
#define    L_CMDERR_L_STAT 0x00800000  /* R/WC PCI error log status      */
/* laerr - R PCI address error log 0x00000000 */
#define    LAERR       0xFFFFFFFF  /* PCI address error log MASK         */
/* dctl - DMA transfer control register */
#define    DCTL_L2V_I  0x00000000  /* R/W direction: VME -> PCI          */
#define    DCTL_L2V_O  0x80000000  /* R/W direction: PCI -> VME          */
#define    DCTL_VDW_08 0x00000000  /* R/W VMEbus max data width D08      */
#define    DCTL_VDW_16 0x00400000  /* R/W VMEbus max data width D16      */
#define    DCTL_VDW_32 0x00800000  /* R/W VMEbus max data width D32      */
#define    DCTL_VDW_64 0x00C00000  /* R/W VMEbus max data width D64      */
#define    DCTL_VAS_16 0x00000000  /* R/W VMEbus address space A16       */
#define    DCTL_VAS_24 0x00010000  /* R/W VMEbus address space A24       */
#define    DCTL_VAS_32 0x00020000  /* R/W VMEbus address space A32       */
#define    DCTL_VAS_R1 0x00030000  /* R/W VMEbus address space reserved 1 */
#define    DCTL_VAS_R2 0x00040000  /* R/W VMEbus address space reserved 2 */
#define    DCTL_VAS_R3 0x00050000  /* R/W VMEbus address space reserved 3 */
#define    DCTL_VAS_U1 0x00060000  /* R/W VMEbus address space user 1     */
#define    DCTL_VAS_U2 0x00070000  /* R/W VMEbus address space user 2     */
#define    DCTL_PGM_D  0x00000000  /* R/W VMEbus data AM code            */
#define    DCTL_PGM_P  0x00004000  /* R/W VMEbus program AM code         */
#define    DCTL_SUPER  0x00001000  /* R/W VMEbus supervisory AM code     */
#define    DCTL_VCT_S  0x00000000  /* R/W VMEbus single cycles only      */
```

```
#define    DCTL_VCT_SB 0x00000100  /* R/W VMEbus single cycles and block    */
#define    DCTL_LD64EN 0x00000080  /* R/W enable 64 bit PCI transaction     */
/* dtbc - DMA transfer byte count register 0xXX000000 */
#define    DTBC        0x00FFFFFF  /* R/W DMA xfer byte count MASK          */
/* dla - DMA PCIbus address register 0x0000000X */
#define    DLA         0xFFFFFFFF  /* R/W DMA PCIbus address MASK           */
/* dva - DMA VMEbus address register 0x0000000X */
#define    DVA         0xFFFFFFFF  /* R/W DMA VMEbus address MASK           */
/* dcpp - DMA command packet pointer 0x0000000X */
#define    DCPP        0xFFFFFFF8  /* R/W DMA command packet pointer MASK   */
/* dgcs - DMA general control/status register */
#define    DGCS_GO         0x80000000  /* R0/W DMA go bit                   */
#define    DGCS_STOP_REQ   0x40000000  /* R0/W DMA stop request             */
#define    DGCS_HALT_REQ   0x20000000  /* R0/W DMA halt request             */
#define    DGCS_CHAIN      0x08000000  /* R/W DMA chaining                  */
#define    DGCS_VON1       0x00000000  /* R/W VME aligned DMA xfer cnt DONE
*/
#define    DGCS_VON2       0x00100000  /* R/W VME aligned DMA xfer cnt 256  */
#define    DGCS_VON3       0x00200000  /* R/W VME aligned DMA xfer cnt 512  */
#define    DGCS_VON4       0x00300000  /* R/W VME aligned DMA xfer cnt 1024 */
#define    DGCS_VON5       0x00400000  /* R/W VME aligned DMA xfer cnt 2048 */
#define    DGCS_VON6       0x00500000  /* R/W VME aligned DMA xfer cnt 4096 */
#define    DGCS_VON7       0x00600000  /* R/W VME aligned DMA xfer cnt 8192 */
#define    DGCS_VON8       0x00700000  /* R/W VME aligned DMA xfer cnt 16384*/
#define    DGCS_VOFF1      0x00000000  /* R/W min off between xfers 0 us    */
#define    DGCS_VOFF2      0x00010000  /* R/W min off between xfers 16 us   */
#define    DGCS_VOFF3      0x00020000  /* R/W min off between xfers 32 us   */
#define    DGCS_VOFF4      0x00030000  /* R/W min off between xfers 64 us   */
#define    DGCS_VOFF5      0x00040000  /* R/W min off between xfers 128 us  */
#define    DGCS_VOFF6      0x00050000  /* R/W min off between xfers 256 us  */
#define    DGCS_VOFF7      0x00060000  /* R/W min off between xfers 512 us  */
#define    DGCS_VOFF8      0x00070000  /* R/W min off between xfers 1024 us */
```

```
#define    DGCS_VOFF9     0x00080000  /* R/W min off between xfers 2 us    */
#define    DGCS_VOFFA     0x00090000  /* R/W min off between xfers 4 us    */
#define    DGCS_VOFFB     0x000A0000  /* R/W min off between xfers 8 us    */
#define    DGCS_ACT       0x00008000  /* R   DMA active flag               */
#define    DGCS_STOP      0x00004000  /* R/WC DMA stopped flag             */
#define    DGCS_HALT      0x00002000  /* R/WC DMA halted flag              */
#define    DGCS_DONE      0x00000800  /* R/WC DMA transfers complete flag  */
#define    DGCS_LERR      0x00000400  /* R/WC DMA PCi bus error            */
#define    DGCS_VERR      0x00000200  /* R/WC DMA VMEbus error             */
#define    DGCS_P_ERR     0x00000100  /* R/WC protocol error               */
#define    DGCS_INT_STOP  0x00000040  /* R/W interrupt when stopped        */
#define    DGCS_INT_HALT  0x00000020  /* R/W interrupt when halted         */
#define    DGCS_INT_DONE  0x00000008  /* R/W interrupt when done           */
#define    DGCS_INT_LERR  0x00000004  /* R/W interrupt on LERR             */
#define    DGCS_INT_VERR  0x00000002  /* R/W interrupt on VERR             */
#define    DGCS_INT_M_ERR 0x00000001  /* R/W interrupt on protocol error   */
/* d_llue - DMA linked list update enable register */
#define    D_LLUE_UPDATE  0x80000000  /* R/W PCI resource updating list    */
/* lint_en - PCI interrupt enable register */
#define    LINT_EN_LM3    0x00800000  /* R/W Location monitor 3 enable     */
#define    LINT_EN_LM2    0x00400000  /* R/W Location monitor 2 enable     */
#define    LINT_EN_LM1    0x00200000  /* R/W Location monitor 1 enable     */
#define    LINT_EN_LM0    0x00100000  /* R/W Location monitor 0 enable     */
#define    LINT_EN_MBOX3  0x00080000  /* R/W MAILBOX 3 enable              */
#define    LINT_EN_MBOX2  0x00040000  /* R/W MAILBOX 2 enable              */
#define    LINT_EN_MBOX1  0x00020000  /* R/W MAILBOX 1 enable              */
#define    LINT_EN_MBOX0  0x00010000  /* R/W MAILBOX 0 enable              */
#define    LINT_EN_ACFAIL 0x00008000  /* R/W ACFAIL interrupt enable       */
#define    LINT_EN_SYSFAIL 0x00004000 /* R/W SYSFAIL interrupt enable      */
#define    LINT_EN_SW_INT 0x00002000  /* R/W PCI software int. enable      */
#define    LINT_EN_SW_IACK 0x00001000 /* R/W VME software IACK enable
*/
```

```
#define   LINT_EN_VERR   0x00000400  /* R/W PCI VERR interrupt enable    */
#define   LINT_EN_LERR   0x00000200  /* R/W PCI LERR interrupt enable    */
#define   LINT_EN_DMA    0x00000100  /* R/W PCI DMA interrupt enable     */
#define   LINT_EN_VIRQ7  0x00000080  /* R/W VIRQ7 interrupt enable       */
#define   LINT_EN_VIRQ6  0x00000040  /* R/W VIRQ6 interrupt enable       */
#define   LINT_EN_VIRQ5  0x00000030  /* R/W VIRQ5 interrupt enable       */
#define   LINT_EN_VIRQ4  0x00000010  /* R/W VIRQ4 interrupt enable       */
#define   LINT_EN_VIRQ3  0x00000008  /* R/W VIRQ3 interrupt enable       */
#define   LINT_EN_VIRQ2  0x00000004  /* R/W VIRQ2 interrupt enable       */
#define   LINT_EN_VIRQ1  0x00000002  /* R/W VIRQ1 interrupt enable       */
#define   LINT_EN_VOWN   0x00000001  /* R/W VOWN interrupt enable        */
/* lint_stat - PCI interrupt status register */
#define   LINT_STAT_LM3   0x00800000 /* R/W Location monitor 3 received  */
#define   LINT_STAT_LM2   0x00400000 /* R/W Location monitor 2 received  */
#define   LINT_STAT_LM1   0x00200000 /* R/W Location monitor 1 received  */
#define   LINT_STAT_LM0   0x00100000 /* R/W Location monitor 0 received  */
#define   LINT_STAT_MBOX3  0x00080000 /* R/W MAILBOX 3 received          */
#define   LINT_STAT_MBOX2  0x00040000 /* R/W MAILBOX 2 received          */
#define   LINT_STAT_MBOX1  0x00020000 /* R/W MAILBOX 1 received          */
#define   LINT_STAT_MBOX0  0x00010000 /* R/W MAILBOX 0 received          */
#define   LINT_STAT_ACFAIL 0x00008000 /* R ACFAIL interrupt active       */
#define   LINT_STAT_SYSFAIL 0x00004000 /* R SYSFAIL interrupt active     */
#define   LINT_STAT_SW_INT 0x00002000 /* R/WC PCI software int. received  */
#define  LINT_STAT_SW_IACK 0x00001000 /* R/WC VME software IACK received
*/
#define   LINT_STAT_VERR   0x00000400 /* R/WC PCI VERR interrupt received */
#define   LINT_STAT_LERR   0x00000200 /* R/WC PCI LERR interrupt received */
#define   LINT_STAT_DMA    0x00000100 /* R/WC PCI DMA interrupt received  */
#define   LINT_STAT_VIRQ7  0x00000080 /* R/WC VIRQ7 interrupt received    */
#define   LINT_STAT_VIRQ6  0x00000040 /* R/WC VIRQ6 interrupt received    */
#define   LINT_STAT_VIRQ5  0x00000030 /* R/WC VIRQ5 interrupt received    */
#define   LINT_STAT_VIRQ4  0x00000010 /* R/WC VIRQ4 interrupt received    */
```

```
#define    LINT_STAT_VIRQ3   0x00000008 /* R/WC VIRQ3 interrupt received    */
#define    LINT_STAT_VIRQ2   0x00000004 /* R/WC VIRQ2 interrupt received    */
#define    LINT_STAT_VIRQ1   0x00000002 /* R/WC VIRQ1 interrupt received    */
#define    LINT_STAT_VOWN    0x00000001 /* R/WC VOWN interrupt received
*/
/* lint_map0 - PCI interrupt map 0 register */
#define    LINT_MAP0_VIRQ7_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_1 0x10000000 /* R/W PCI int LINT#1 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_2 0x20000000 /* R/W PCI int LINT#2 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_3 0x30000000 /* R/W PCI int LINT#3 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_4 0x40000000 /* R/W PCI int LINT#4 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_5 0x50000000 /* R/W PCI int LINT#5 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_6 0x60000000 /* R/W PCI int LINT#6 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ7_7 0x70000000 /* R/W PCI int LINT#7 for VME IRQ7
*/
#define    LINT_MAP0_VIRQ6_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_1 0x01000000 /* R/W PCI int LINT#1 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_2 0x02000000 /* R/W PCI int LINT#2 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_3 0x03000000 /* R/W PCI int LINT#3 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_4 0x04000000 /* R/W PCI int LINT#4 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_5 0x05000000 /* R/W PCI int LINT#5 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_6 0x06000000 /* R/W PCI int LINT#6 for VME IRQ6
*/
#define    LINT_MAP0_VIRQ6_7 0x07000000 /* R/W PCI int LINT#7 for VME IRQ6
*/
```

```
#define    LINT_MAP0_VIRQ5_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_1 0x00100000 /* R/W PCI int LINT#1 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_2 0x00200000 /* R/W PCI int LINT#2 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_3 0x00300000 /* R/W PCI int LINT#3 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_4 0x00400000 /* R/W PCI int LINT#4 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_5 0x00500000 /* R/W PCI int LINT#5 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_6 0x00600000 /* R/W PCI int LINT#6 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ5_7 0x00700000 /* R/W PCI int LINT#7 for VME IRQ5
*/

#define    LINT_MAP0_VIRQ4_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_1 0x00010000 /* R/W PCI int LINT#1 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_2 0x00020000 /* R/W PCI int LINT#2 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_3 0x00030000 /* R/W PCI int LINT#3 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_4 0x00040000 /* R/W PCI int LINT#4 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_5 0x00050000 /* R/W PCI int LINT#5 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_6 0x00060000 /* R/W PCI int LINT#6 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ4_7 0x00070000 /* R/W PCI int LINT#7 for VME IRQ4
*/

#define    LINT_MAP0_VIRQ3_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_1 0x00001000 /* R/W PCI int LINT#1 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_2 0x00002000 /* R/W PCI int LINT#2 for VME IRQ3
*/
```

```
#define    LINT_MAP0_VIRQ3_3 0x00003000 /* R/W PCI int LINT#3 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_4 0x00004000 /* R/W PCI int LINT#4 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_5 0x00005000 /* R/W PCI int LINT#5 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_6 0x00006000 /* R/W PCI int LINT#6 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ3_7 0x00007000 /* R/W PCI int LINT#7 for VME IRQ3
*/

#define    LINT_MAP0_VIRQ2_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_1 0x00000100 /* R/W PCI int LINT#1 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_2 0x00000200 /* R/W PCI int LINT#2 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_3 0x00000300 /* R/W PCI int LINT#3 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_4 0x00000400 /* R/W PCI int LINT#4 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_5 0x00000500 /* R/W PCI int LINT#5 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_6 0x00000600 /* R/W PCI int LINT#6 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ2_7 0x00000700 /* R/W PCI int LINT#7 for VME IRQ2
*/

#define    LINT_MAP0_VIRQ1_0 0x00000000 /* R/W PCI int LINT#0 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_1 0x00000010 /* R/W PCI int LINT#1 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_2 0x00000020 /* R/W PCI int LINT#2 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_3 0x00000030 /* R/W PCI int LINT#3 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_4 0x00000040 /* R/W PCI int LINT#4 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_5 0x00000050 /* R/W PCI int LINT#5 for VME IRQ1
*/
```

```
#define    LINT_MAP0_VIRQ1_6 0x00000060 /* R/W PCI int LINT#6 for VME IRQ1
*/

#define    LINT_MAP0_VIRQ1_7 0x00000070 /* R/W PCI int LINT#7 for VME IRQ1
*/

#define    LINT_MAP0_VOWN_0  0x00000000 /* R/W PCI int LINT#0 for VME
OWN   */

#define    LINT_MAP0_VOWN_1  0x00000001 /* R/W PCI int LINT#1 for VME
OWN   */

#define    LINT_MAP0_VOWN_2  0x00000002 /* R/W PCI int LINT#2 for VME
OWN   */

#define    LINT_MAP0_VOWN_3  0x00000003 /* R/W PCI int LINT#3 for VME
OWN   */

#define    LINT_MAP0_VOWN_4  0x00000004 /* R/W PCI int LINT#4 for VME
OWN   */

#define    LINT_MAP0_VOWN_5  0x00000005 /* R/W PCI int LINT#5 for VME
OWN   */

#define    LINT_MAP0_VOWN_6  0x00000006 /* R/W PCI int LINT#6 for VME
OWN   */

#define    LINT_MAP0_VOWN_7  0x00000007 /* R/W PCI int LINT#7 for VME
OWN   */

/* lint_map1 - PCI interrupt map 1 register */

#define    LINT_MAP1_ACFAIL_0  0x00000000 /* R/W PCI int LINT#0 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_1  0x10000000 /* R/W PCI int LINT#1 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_2  0x20000000 /* R/W PCI int LINT#2 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_3  0x30000000 /* R/W PCI int LINT#3 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_4  0x40000000 /* R/W PCI int LINT#4 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_5  0x50000000 /* R/W PCI int LINT#5 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_6  0x60000000 /* R/W PCI int LINT#6 for ACFAIL
*/

#define    LINT_MAP1_ACFAIL_7  0x70000000 /* R/W PCI int LINT#7 for ACFAIL
*/
```

```
#define    LINT_MAP1_SYSFAIL_0 0x00000000 /* R/W PCI int LINT#0 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_1 0x01000000 /* R/W PCI int LINT#1 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_2 0x02000000 /* R/W PCI int LINT#2 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_3 0x03000000 /* R/W PCI int LINT#3 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_4 0x04000000 /* R/W PCI int LINT#4 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_5 0x05000000 /* R/W PCI int LINT#5 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_6 0x06000000 /* R/W PCI int LINT#6 for SYSFAIL
*/

#define    LINT_MAP1_SYSFAIL_7 0x07000000 /* R/W PCI int LINT#7 for SYSFAIL
*/

#define    LINT_MAP1_SW_INT_0  0x00000000 /* R/W PCI int LINT#0 for SW_INT
*/

#define    LINT_MAP1_SW_INT_1  0x00100000 /* R/W PCI int LINT#1 for SW_INT
*/

#define    LINT_MAP1_SW_INT_2  0x00200000 /* R/W PCI int LINT#2 for SW_INT
*/

#define    LINT_MAP1_SW_INT_3  0x00300000 /* R/W PCI int LINT#3 for SW_INT
*/

#define    LINT_MAP1_SW_INT_4  0x00400000 /* R/W PCI int LINT#4 for SW_INT
*/

#define    LINT_MAP1_SW_INT_5  0x00500000 /* R/W PCI int LINT#5 for SW_INT
*/

#define    LINT_MAP1_SW_INT_6  0x00600000 /* R/W PCI int LINT#6 for SW_INT
*/

#define    LINT_MAP1_SW_INT_7  0x00700000 /* R/W PCI int LINT#7 for SW_INT
*/

#define    LINT_MAP1_SW_IACK_0 0x00000000 /* R/W PCI int LINT#0 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_1 0x00010000 /* R/W PCI int LINT#1 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_2 0x00020000 /* R/W PCI int LINT#2 for
SW_IACK */
```

```
#define    LINT_MAP1_SW_IACK_3 0x00030000 /* R/W PCI int LINT#3 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_4 0x00040000 /* R/W PCI int LINT#4 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_5 0x00050000 /* R/W PCI int LINT#5 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_6 0x00060000 /* R/W PCI int LINT#6 for
SW_IACK */

#define    LINT_MAP1_SW_IACK_7 0x00070000 /* R/W PCI int LINT#7 for
SW_IACK */
#define    LINT_MAP1_VERR_0    0x00000000 /* R/W PCI int LINT#0 for VERR    */
#define    LINT_MAP1_VERR_1    0x00000100 /* R/W PCI int LINT#1 for VERR    */
#define    LINT_MAP1_VERR_2    0x00000200 /* R/W PCI int LINT#2 for VERR    */
#define    LINT_MAP1_VERR_3    0x00000300 /* R/W PCI int LINT#3 for VERR    */
#define    LINT_MAP1_VERR_4    0x00000400 /* R/W PCI int LINT#4 for VERR    */
#define    LINT_MAP1_VERR_5    0x00000500 /* R/W PCI int LINT#5 for VERR    */
#define    LINT_MAP1_VERR_6    0x00000600 /* R/W PCI int LINT#6 for VERR    */
#define    LINT_MAP1_VERR_7    0x00000700 /* R/W PCI int LINT#7 for VERR    */
#define    LINT_MAP1_LERR_0    0x00000000 /* R/W PCI int LINT#0 for LERR    */
#define    LINT_MAP1_LERR_1    0x00000010 /* R/W PCI int LINT#1 for LERR    */
#define    LINT_MAP1_LERR_2    0x00000020 /* R/W PCI int LINT#2 for LERR    */
#define    LINT_MAP1_LERR_3    0x00000030 /* R/W PCI int LINT#3 for LERR    */
#define    LINT_MAP1_LERR_4    0x00000040 /* R/W PCI int LINT#4 for LERR    */
#define    LINT_MAP1_LERR_5    0x00000050 /* R/W PCI int LINT#5 for LERR    */
#define    LINT_MAP1_LERR_6    0x00000060 /* R/W PCI int LINT#6 for LERR    */
#define    LINT_MAP1_LERR_7    0x00000070 /* R/W PCI int LINT#7 for LERR    */
#define    LINT_MAP1_DMA_0     0x00000000 /* R/W PCI int LINT#0 for DMA     */
#define    LINT_MAP1_DMA_1     0x00000001 /* R/W PCI int LINT#1 for DMA     */
#define    LINT_MAP1_DMA_2     0x00000002 /* R/W PCI int LINT#2 for DMA     */
#define    LINT_MAP1_DMA_3     0x00000003 /* R/W PCI int LINT#3 for DMA     */
#define    LINT_MAP1_DMA_4     0x00000004 /* R/W PCI int LINT#4 for DMA     */
#define    LINT_MAP1_DMA_5     0x00000005 /* R/W PCI int LINT#5 for DMA     */
#define    LINT_MAP1_DMA_6     0x00000006 /* R/W PCI int LINT#6 for DMA     */
#define    LINT_MAP1_DMA_7     0x00000007 /* R/W PCI int LINT#7 for DMA     */
```

```
/* vint_en - VMEbus interrupt enable register */
#define   VINT_EN_SW7     0x80000000 /* R/W enable VMEbus int SW7      */
#define   VINT_EN_SW6     0x40000000 /* R/W enable VMEbus int SW6      */
#define   VINT_EN_SW5     0x20000000 /* R/W enable VMEbus int SW5      */
#define   VINT_EN_SW4     0x10000000 /* R/W enable VMEbus int SW4      */
#define   VINT_EN_SW3     0x08000000 /* R/W enable VMEbus int SW3      */
#define   VINT_EN_SW2     0x04000000 /* R/W enable VMEbus int SW2      */
#define   VINT_EN_SW1     0x02000000 /* R/W enable VMEbus int SW1      */
#define   VINT_EN_MBOX3   0x00080000 /* R/W enable VMEbus int MAILBOX 3
*/
#define   VINT_EN_MBOX2   0x00040000 /* R/W enable VMEbus int MAILBOX 2
*/
#define   VINT_EN_MBOX1   0x00020000 /* R/W enable VMEbus int MAILBOX 1
*/
#define   VINT_EN_MBOX0   0x00010000 /* R/W enable VMEbus int MAILBOX 0
*/
#define   VINT_EN_SW_IACK 0x00001000 /* R/W enable VMEbus int SW_IACK
*/
#define   VINT_EN_VERR    0x00000400 /* R/W enable PCIbus int VERR     */
#define   VINT_EN_LERR    0x00000200 /* R/W enable PCIbus int LERR     */
#define   VINT_EN_DMA     0x00000100 /* R/W enable PCIbus int DMA      */
#define   VINT_EN_LINT7   0x00000080 /* R/W enable PCIbus int LINT7    */
#define   VINT_EN_LINT6   0x00000040 /* R/W enable PCIbus int LINT6    */
#define   VINT_EN_LINT5   0x00000020 /* R/W enable PCIbus int LINT5    */
#define   VINT_EN_LINT4   0x00000010 /* R/W enable PCIbus int LINT4    */
#define   VINT_EN_LINT3   0x00000008 /* R/W enable PCIbus int LINT3    */
#define   VINT_EN_LINT2   0x00000004 /* R/W enable PCIbus int LINT2    */
#define   VINT_EN_LINT1   0x00000002 /* R/W enable PCIbus int LINT1    */
#define   VINT_EN_LINT0   0x00000001 /* R/W enable PCIbus int LINT0    */
/* vint_stat - VMEbus interrupt status register */
#define   VINT_STAT_SW7   0x80000000 /* R/W  VMEbus int SW7        */
#define   VINT_STAT_SW6   0x40000000 /* R/W  VMEbus int SW6        */
#define   VINT_STAT_SW5   0x20000000 /* R/W  VMEbus int SW5        */
#define   VINT_STAT_SW4   0x10000000 /* R/W  VMEbus int SW4        */
```

```
#define    VINT_STAT_SW3    0x08000000 /* R/W  VMEbus int SW3          */
#define    VINT_STAT_SW2    0x04000000 /* R/W  VMEbus int SW2          */
#define    VINT_STAT_SW1    0x02000000 /* R/W  VMEbus int SW1          */
#define    VINT_STAT_MBOX3  0x00080000 /* R/W  VMEbus int MAILBOX 3    */
#define    VINT_STAT_MBOX2  0x00040000 /* R/W  VMEbus int MAILBOX 2    */
#define    VINT_STAT_MBOX1  0x00020000 /* R/W  VMEbus int MAILBOX 1    */
#define    VINT_STAT_MBOX0  0x00010000 /* R/W  VMEbus int MAILBOX 0    */
#define    VINT_STAT_SW_IACK 0x00001000 /* R/WC VMEbus int SW_IACK
*/
#define    VINT_STAT_VERR   0x00000400 /* R/WC VMEbus int VERR         */
#define    VINT_STAT_LERR   0x00000200 /* R/WC VMEbus int LERR         */
#define    VINT_STAT_DMA    0x00000100 /* R/WC VMEbus int DMA          */
#define    VINT_STAT_LINT7  0x00000080 /* R/WC VMEbus int LINT7        */
#define    VINT_STAT_LINT6  0x00000040 /* R/WC VMEbus int LINT6        */
#define    VINT_STAT_LINT5  0x00000020 /* R/WC VMEbus int LINT5        */
#define    VINT_STAT_LINT4  0x00000010 /* R/WC VMEbus int LINT4        */
#define    VINT_STAT_LINT3  0x00000008 /* R/WC VMEbus int LINT3        */
#define    VINT_STAT_LINT2  0x00000004 /* R/WC VMEbus int LINT2        */
#define    VINT_STAT_LINT1  0x00000002 /* R/WC VMEbus int LINT1        */
#define    VINT_STAT_LINT0  0x00000001 /* R/WC VMEbus int LINT0        */
/* vint_map0 - VME interrupt map 0 register */
#define    VINT_MAP0_LINT7_D 0x00000000 /* R/W VME int disable for LINT7
*/
#define    VINT_MAP0_LINT7_1 0x10000000 /* R/W VME int 1 for LINT7      */
#define    VINT_MAP0_LINT7_2 0x20000000 /* R/W VME int 2 for LINT7      */
#define    VINT_MAP0_LINT7_3 0x30000000 /* R/W VME int 3 for LINT7      */
#define    VINT_MAP0_LINT7_4 0x40000000 /* R/W VME int 4 for LINT7      */
#define    VINT_MAP0_LINT7_5 0x50000000 /* R/W VME int 5 for LINT7      */
#define    VINT_MAP0_LINT7_6 0x60000000 /* R/W VME int 6 for LINT7      */
#define    VINT_MAP0_LINT7_7 0x70000000 /* R/W VME int 7 for LINT7      */
#define    VINT_MAP0_LINT6_D 0x00000000 /* R/W VME int disable for LINT6
*/
#define    VINT_MAP0_LINT6_1 0x01000000 /* R/W VME int 1 for LINT6      */
```

```
#define    VINT_MAP0_LINT6_2 0x02000000 /* R/W VME int 2 for LINT6        */
#define    VINT_MAP0_LINT6_3 0x03000000 /* R/W VME int 3 for LINT6        */
#define    VINT_MAP0_LINT6_4 0x04000000 /* R/W VME int 4 for LINT6        */
#define    VINT_MAP0_LINT6_5 0x05000000 /* R/W VME int 5 for LINT6        */
#define    VINT_MAP0_LINT6_6 0x06000000 /* R/W VME int 6 for LINT6        */
#define    VINT_MAP0_LINT6_7 0x07000000 /* R/W VME int 7 for LINT6        */
#define    VINT_MAP0_LINT5_D 0x00000000 /* R/W VME int disable for LINT5
*/
#define    VINT_MAP0_LINT5_1 0x00100000 /* R/W VME int 1 for LINT5        */
#define    VINT_MAP0_LINT5_2 0x00200000 /* R/W VME int 2 for LINT5        */
#define    VINT_MAP0_LINT5_3 0x00300000 /* R/W VME int 3 for LINT5        */
#define    VINT_MAP0_LINT5_4 0x00400000 /* R/W VME int 4 for LINT5        */
#define    VINT_MAP0_LINT5_5 0x00500000 /* R/W VME int 5 for LINT5        */
#define    VINT_MAP0_LINT5_6 0x00600000 /* R/W VME int 6 for LINT5        */
#define    VINT_MAP0_LINT5_7 0x00700000 /* R/W VME int 7 for LINT5        */
#define    VINT_MAP0_LINT4_D 0x00000000 /* R/W VME int disable for LINT4
*/
#define    VINT_MAP0_LINT4_1 0x00010000 /* R/W VME int 1 for LINT4        */
#define    VINT_MAP0_LINT4_2 0x00020000 /* R/W VME int 2 for LINT4        */
#define    VINT_MAP0_LINT4_3 0x00030000 /* R/W VME int 3 for LINT4        */
#define    VINT_MAP0_LINT4_4 0x00040000 /* R/W VME int 4 for LINT4        */
#define    VINT_MAP0_LINT4_5 0x00050000 /* R/W VME int 5 for LINT4        */
#define    VINT_MAP0_LINT4_6 0x00060000 /* R/W VME int 6 for LINT4        */
#define    VINT_MAP0_LINT4_7 0x00070000 /* R/W VME int 7 for LINT4        */
#define    VINT_MAP0_LINT3_D 0x00000000 /* R/W VME int disable for LINT3
*/
#define    VINT_MAP0_LINT3_1 0x00001000 /* R/W VME int 1 for LINT3        */
#define    VINT_MAP0_LINT3_2 0x00002000 /* R/W VME int 2 for LINT3        */
#define    VINT_MAP0_LINT3_3 0x00003000 /* R/W VME int 3 for LINT3        */
#define    VINT_MAP0_LINT3_4 0x00004000 /* R/W VME int 4 for LINT3        */
#define    VINT_MAP0_LINT3_5 0x00005000 /* R/W VME int 5 for LINT3        */
#define    VINT_MAP0_LINT3_6 0x00006000 /* R/W VME int 6 for LINT3        */
#define    VINT_MAP0_LINT3_7 0x00007000 /* R/W VME int 7 for LINT3        */
```

```
#define    VINT_MAP0_LINT2_D 0x00000000 /* R/W VME int disable for LINT2
*/
#define    VINT_MAP0_LINT2_1 0x00000100 /* R/W VME int 1 for LINT2      */
#define    VINT_MAP0_LINT2_2 0x00000200 /* R/W VME int 2 for LINT2      */
#define    VINT_MAP0_LINT2_3 0x00000300 /* R/W VME int 3 for LINT2      */
#define    VINT_MAP0_LINT2_4 0x00000400 /* R/W VME int 4 for LINT2      */
#define    VINT_MAP0_LINT2_5 0x00000500 /* R/W VME int 5 for LINT2      */
#define    VINT_MAP0_LINT2_6 0x00000600 /* R/W VME int 6 for LINT2      */
#define    VINT_MAP0_LINT2_7 0x00000700 /* R/W VME int 7 for LINT2      */
#define    VINT_MAP0_LINT1_D 0x00000000 /* R/W VME int disable for LINT1
*/
#define    VINT_MAP0_LINT1_1 0x00000010 /* R/W VME int 1 for LINT1      */
#define    VINT_MAP0_LINT1_2 0x00000020 /* R/W VME int 2 for LINT1      */
#define    VINT_MAP0_LINT1_3 0x00000030 /* R/W VME int 3 for LINT1      */
#define    VINT_MAP0_LINT1_4 0x00000040 /* R/W VME int 4 for LINT1      */
#define    VINT_MAP0_LINT1_5 0x00000050 /* R/W VME int 5 for LINT1      */
#define    VINT_MAP0_LINT1_6 0x00000060 /* R/W VME int 6 for LINT1      */
#define    VINT_MAP0_LINT1_7 0x00000070 /* R/W VME int 7 for LINT1      */
#define    VINT_MAP0_LINT0_D 0x00000000 /* R/W VME int disable for LINT0
*/
#define    VINT_MAP0_LINT0_1 0x00000001 /* R/W VME int 1 for LINT0      */
#define    VINT_MAP0_LINT0_2 0x00000002 /* R/W VME int 2 for LINT0      */
#define    VINT_MAP0_LINT0_3 0x00000003 /* R/W VME int 3 for LINT0      */
#define    VINT_MAP0_LINT0_4 0x00000004 /* R/W VME int 4 for LINT0      */
#define    VINT_MAP0_LINT0_5 0x00000005 /* R/W VME int 5 for LINT0      */
#define    VINT_MAP0_LINT0_6 0x00000006 /* R/W VME int 6 for LINT0      */
#define    VINT_MAP0_LINT0_7 0x00000007 /* R/W VME int 7 for LINT0      */
/* vint_map1 - VME interrupt map 1 register */
#define    VINT_MAP1_SW_IACK_D 0x00000000 /* R/W VME int disable for
SW_IACK */
#define    VINT_MAP1_SW_IACK_1 0x00010000 /* R/W VME int 1 for SW_IACK
*/
#define    VINT_MAP1_SW_IACK_2 0x00020000 /* R/W VME int 2 for SW_IACK
*/
```

```
#define   VINT_MAP1_SW_IACK_3 0x00030000 /* R/W VME int 3 for SW_IACK
*/
#define   VINT_MAP1_SW_IACK_4 0x00040000 /* R/W VME int 4 for SW_IACK
*/
#define   VINT_MAP1_SW_IACK_5 0x00050000 /* R/W VME int 5 for SW_IACK
*/
#define   VINT_MAP1_SW_IACK_6 0x00060000 /* R/W VME int 6 for SW_IACK
*/
#define   VINT_MAP1_SW_IACK_7 0x00070000 /* R/W VME int 7 for SW_IACK
*/
#define   VINT_MAP1_VERR_D   0x00000000 /* R/W VME int disable for VERR
*/
#define   VINT_MAP1_VERR_1   0x00000100 /* R/W VME int 1 for VERR      */
#define   VINT_MAP1_VERR_2   0x00000200 /* R/W VME int 2 for VERR      */
#define   VINT_MAP1_VERR_3   0x00000300 /* R/W VME int 3 for VERR      */
#define   VINT_MAP1_VERR_4   0x00000400 /* R/W VME int 4 for VERR      */
#define   VINT_MAP1_VERR_5   0x00000500 /* R/W VME int 5 for VERR      */
#define   VINT_MAP1_VERR_6   0x00000600 /* R/W VME int 6 for VERR      */
#define   VINT_MAP1_VERR_7   0x00000700 /* R/W VME int 7 for VERR      */
#define   VINT_MAP1_LERR_D   0x00000000 /* R/W VME int disable for LERR  */
#define   VINT_MAP1_LERR_1   0x00000010 /* R/W VME int 1 for LERR      */
#define   VINT_MAP1_LERR_2   0x00000020 /* R/W VME int 2 for LERR      */
#define   VINT_MAP1_LERR_3   0x00000030 /* R/W VME int 3 for LERR      */
#define   VINT_MAP1_LERR_4   0x00000040 /* R/W VME int 4 for LERR      */
#define   VINT_MAP1_LERR_5   0x00000050 /* R/W VME int 5 for LERR      */
#define   VINT_MAP1_LERR_6   0x00000060 /* R/W VME int 6 for LERR      */
#define   VINT_MAP1_LERR_7   0x00000070 /* R/W VME int 7 for LERR      */
#define   VINT_MAP1_DMA_D   0x00000000 /* R/W VME int disable for LERR
*/
#define   VINT_MAP1_DMA_1   0x00000001 /* R/W VME int 1 for DMA       */
#define   VINT_MAP1_DMA_2   0x00000002 /* R/W VME int 2 for DMA       */
#define   VINT_MAP1_DMA_3   0x00000003 /* R/W VME int 3 for DMA       */
#define   VINT_MAP1_DMA_4   0x00000004 /* R/W VME int 4 for DMA       */
#define   VINT_MAP1_DMA_5   0x00000005 /* R/W VME int 5 for DMA       */
```

```
#define    VINT_MAP1_DMA_6    0x00000006 /* R/W VME int 6 for DMA        */
#define    VINT_MAP1_DMA_7    0x00000007 /* R/W VME int 7 for DMA        */
/* statid - interrupt STATUS/ID OUT 0x00XXXXXX */
#define    STATID        0xFF000000   /* R/W interrupt status/ID out MASK */
/* v1_statid - R VIRQ1 STATUS/ID register 0xXXXXXX00 */
/* v2_statid - R VIRQ2 STATUS/ID register 0xXXXXXX00 */
/* v3_statid - R VIRQ3 STATUS/ID register 0xXXXXXX00 */
/* v4_statid - R VIRQ4 STATUS/ID register 0xXXXXXX00 */
/* v5_statid - R VIRQ5 STATUS/ID register 0xXXXXXX00 */
/* v6_statid - R VIRQ6 STATUS/ID register 0xXXXXXX00 */
/* v7_statid - R VIRQ7 STATUS/ID register 0xXXXXXX00 */
#define    VX_STATID_ERR  0x00000100  /* R VME BERR* occurred during IACK  */
#define    VX_STATID_ID   0x000000FF  /* R VME status/ID MASK          */
/* lint_map2 - local interrupt Map 2 register */
#define    LINT_MAP2_LM3_0    0x00000000 /* R/W PCI int LINT#0 for LOC MON3
*/
#define    LINT_MAP2_LM3_1    0x10000000 /* R/W PCI int LINT#1 for LOC MON3
*/
#define    LINT_MAP2_LM3_2    0x20000000 /* R/W PCI int LINT#2 for LOC MON3
*/
#define    LINT_MAP2_LM3_3    0x30000000 /* R/W PCI int LINT#3 for LOC MON3
*/
#define    LINT_MAP2_LM3_4    0x40000000 /* R/W PCI int LINT#4 for LOC MON3
*/
#define    LINT_MAP2_LM3_5    0x50000000 /* R/W PCI int LINT#5 for LOC MON3
*/
#define    LINT_MAP2_LM3_6    0x60000000 /* R/W PCI int LINT#6 for LOC MON3
*/
#define    LINT_MAP2_LM3_7    0x70000000 /* R/W PCI int LINT#7 for LOC MON3
*/
#define    LINT_MAP2_LM2_0    0x00000000 /* R/W PCI int LINT#0 for LOC MON2
*/
#define    LINT_MAP2_LM2_1    0x01000000 /* R/W PCI int LINT#1 for LOC MON2
*/
#define    LINT_MAP2_LM2_2    0x02000000 /* R/W PCI int LINT#2 for LOC MON2
*/
```

```
#define   LINT_MAP2_LM2_3   0x03000000 /* R/W PCI int LINT#3 for LOC MON2
*/

#define   LINT_MAP2_LM2_4   0x04000000 /* R/W PCI int LINT#4 for LOC MON2
*/

#define   LINT_MAP2_LM2_5   0x05000000 /* R/W PCI int LINT#5 for LOC MON2
*/

#define   LINT_MAP2_LM2_6   0x06000000 /* R/W PCI int LINT#6 for LOC MON2
*/

#define   LINT_MAP2_LM2_7   0x07000000 /* R/W PCI int LINT#7 for LOC MON2
*/

#define   LINT_MAP2_LM1_0   0x00000000 /* R/W PCI int LINT#0 for LOC MON1
*/

#define   LINT_MAP2_LM1_1   0x00100000 /* R/W PCI int LINT#1 for LOC MON1
*/

#define   LINT_MAP2_LM1_2   0x00200000 /* R/W PCI int LINT#2 for LOC MON1
*/

#define   LINT_MAP2_LM1_3   0x00300000 /* R/W PCI int LINT#3 for LOC MON1
*/

#define   LINT_MAP2_LM1_4   0x00400000 /* R/W PCI int LINT#4 for LOC MON1
*/

#define   LINT_MAP2_LM1_5   0x00500000 /* R/W PCI int LINT#5 for LOC MON1
*/

#define   LINT_MAP2_LM1_6   0x00600000 /* R/W PCI int LINT#6 for LOC MON1
*/

#define   LINT_MAP2_LM1_7   0x00700000 /* R/W PCI int LINT#7 for LOC MON1
*/

#define   LINT_MAP2_LM0_0   0x00000000 /* R/W PCI int LINT#0 for
LOC_MON0 */

#define   LINT_MAP2_LM0_1   0x00010000 /* R/W PCI int LINT#1 for
LOC_MON0 */

#define   LINT_MAP2_LM0_2   0x00020000 /* R/W PCI int LINT#2 for
LOC_MON0 */

#define   LINT_MAP2_LM0_3   0x00030000 /* R/W PCI int LINT#3 for
LOC_MON0 */

#define   LINT_MAP2_LM0_4   0x00040000 /* R/W PCI int LINT#4 for
LOC_MON0 */

#define   LINT_MAP2_LM0_5   0x00050000 /* R/W PCI int LINT#5 for
LOC_MON0 */
```

```
#define    LINT_MAP2_LM0_6    0x00060000 /* R/W PCI int LINT#6 for
LOC_MON0 */

#define    LINT_MAP2_LM0_7    0x00070000 /* R/W PCI int LINT#7 for
LOC_MON0 */

#define    LINT_MAP2_MB3_0    0x00000000 /* R/W PCI int LINT#0 for MAILBOX3
*/

#define    LINT_MAP2_MB3_1    0x00001000 /* R/W PCI int LINT#1 for MAILBOX3
*/

#define    LINT_MAP2_MB3_2    0x00002000 /* R/W PCI int LINT#2 for MAILBOX3
*/

#define    LINT_MAP2_MB3_3    0x00003000 /* R/W PCI int LINT#3 for MAILBOX3
*/

#define    LINT_MAP2_MB3_4    0x00004000 /* R/W PCI int LINT#4 for MAILBOX3
*/

#define    LINT_MAP2_MB3_5    0x00005000 /* R/W PCI int LINT#5 for MAILBOX3
*/

#define    LINT_MAP2_MB3_6    0x00006000 /* R/W PCI int LINT#6 for MAILBOX3
*/

#define    LINT_MAP2_MB3_7    0x00007000 /* R/W PCI int LINT#7 for MAILBOX3
*/

#define    LINT_MAP2_MB2_0    0x00000000 /* R/W PCI int LINT#0 for MAILBOX2
*/

#define    LINT_MAP2_MB2_1    0x00000100 /* R/W PCI int LINT#1 for MAILBOX2
*/

#define    LINT_MAP2_MB2_2    0x00000200 /* R/W PCI int LINT#2 for MAILBOX2
*/

#define    LINT_MAP2_MB2_3    0x00000300 /* R/W PCI int LINT#3 for MAILBOX2
*/

#define    LINT_MAP2_MB2_4    0x00000400 /* R/W PCI int LINT#4 for MAILBOX2
*/

#define    LINT_MAP2_MB2_5    0x00000500 /* R/W PCI int LINT#5 for MAILBOX2
*/

#define    LINT_MAP2_MB2_6    0x00000600 /* R/W PCI int LINT#6 for MAILBOX2
*/

#define    LINT_MAP2_MB2_7    0x00000700 /* R/W PCI int LINT#7 for MAILBOX2
*/

#define    LINT_MAP2_MB1_0    0x00000000 /* R/W PCI int LINT#0 for MAILBOX1
*/
```

```
#define    LINT_MAP2_MB1_1    0x00000010 /* R/W PCI int LINT#1 for MAILBOX1
*/

#define    LINT_MAP2_MB1_2    0x00000020 /* R/W PCI int LINT#2 for MAILBOX1
*/

#define    LINT_MAP2_MB1_3    0x00000030 /* R/W PCI int LINT#3 for MAILBOX1
*/

#define    LINT_MAP2_MB1_4    0x00000040 /* R/W PCI int LINT#4 for MAILBOX1
*/

#define    LINT_MAP2_MB1_5    0x00000050 /* R/W PCI int LINT#5 for MAILBOX1
*/

#define    LINT_MAP2_MB1_6    0x00000060 /* R/W PCI int LINT#6 for MAILBOX1
*/

#define    LINT_MAP2_MB1_7    0x00000070 /* R/W PCI int LINT#7 for MAILBOX1
*/

#define    LINT_MAP2_MB0_0    0x00000000 /* R/W PCI int LINT#0 for MAILBOX0
*/

#define    LINT_MAP2_MB0_1    0x00000001 /* R/W PCI int LINT#1 for MAILBOX0
*/

#define    LINT_MAP2_MB0_2    0x00000002 /* R/W PCI int LINT#2 for MAILBOX0
*/

#define    LINT_MAP2_MB0_3    0x00000003 /* R/W PCI int LINT#3 for MAILBOX0
*/

#define    LINT_MAP2_MB0_4    0x00000004 /* R/W PCI int LINT#4 for MAILBOX0
*/

#define    LINT_MAP2_MB0_5    0x00000005 /* R/W PCI int LINT#5 for MAILBOX0
*/

#define    LINT_MAP2_MB0_6    0x00000006 /* R/W PCI int LINT#6 for MAILBOX0
*/

#define    LINT_MAP2_MB0_7    0x00000007 /* R/W PCI int LINT#7 for MAILBOX0
*/

/* vint_map2 - vme interrupt Map 2 register */

#define    VINT_MAP2_MB3_1    0x00001000 /* R/W VME int VIRQ#1 for
MAILBOX3 */

#define    VINT_MAP2_MB3_2    0x00002000 /* R/W VME int VIRQ#2 for
MAILBOX3 */

#define    VINT_MAP2_MB3_3    0x00003000 /* R/W VME int VIRQ#3 for
MAILBOX3 */
```

```
#define VINT_MAP2_MB3_4 0x00004000 /* R/W VME int VIRQ#4 for MAILBOX3 */
#define VINT_MAP2_MB3_5 0x00005000 /* R/W VME int VIRQ#5 for MAILBOX3 */
#define VINT_MAP2_MB3_6 0x00006000 /* R/W VME int VIRQ#6 for MAILBOX3 */
#define VINT_MAP2_MB3_7 0x00007000 /* R/W VME int VIRQ#7 for MAILBOX3 */
#define VINT_MAP2_MB2_1 0x00000100 /* R/W VME int VIRQ#1 for MAILBOX2 */
#define VINT_MAP2_MB2_2 0x00000200 /* R/W VME int VIRQ#2 for MAILBOX2 */
#define VINT_MAP2_MB2_3 0x00000300 /* R/W VME int VIRQ#3 for MAILBOX2 */
#define VINT_MAP2_MB2_4 0x00000400 /* R/W VME int VIRQ#4 for MAILBOX2 */
#define VINT_MAP2_MB2_5 0x00000500 /* R/W VME int VIRQ#5 for MAILBOX2 */
#define VINT_MAP2_MB2_6 0x00000600 /* R/W VME int VIRQ#6 for MAILBOX2 */
#define VINT_MAP2_MB2_7 0x00000700 /* R/W VME int VIRQ#7 for MAILBOX2 */
#define VINT_MAP2_MB1_1 0x00000010 /* R/W VME int VIRQ#1 for MAILBOX1 */
#define VINT_MAP2_MB1_2 0x00000020 /* R/W VME int VIRQ#2 for MAILBOX1 */
#define VINT_MAP2_MB1_3 0x00000030 /* R/W VME int VIRQ#3 for MAILBOX1 */
#define VINT_MAP2_MB1_4 0x00000040 /* R/W VME int VIRQ#4 for MAILBOX1 */
#define VINT_MAP2_MB1_5 0x00000050 /* R/W VME int VIRQ#5 for MAILBOX1 */
#define VINT_MAP2_MB1_6 0x00000060 /* R/W VME int VIRQ#6 for MAILBOX1 */
#define VINT_MAP2_MB1_7 0x00000070 /* R/W VME int VIRQ#7 for MAILBOX1 */
#define VINT_MAP2_MB0_1 0x00000001 /* R/W VME int VIRQ#1 for MAILBOX0 */
```

```
#define    VINT_MAP2_MB0_2    0x00000002 /* R/W VME int VIRQ#2 for
MAILBOX0 */

#define    VINT_MAP2_MB0_3    0x00000003 /* R/W VME int VIRQ#3 for
MAILBOX0 */

#define    VINT_MAP2_MB0_4    0x00000004 /* R/W VME int VIRQ#4 for
MAILBOX0 */

#define    VINT_MAP2_MB0_5    0x00000005 /* R/W VME int VIRQ#5 for
MAILBOX0 */

#define    VINT_MAP2_MB0_6    0x00000006 /* R/W VME int VIRQ#6 for
MAILBOX0 */

#define    VINT_MAP2_MB0_7    0x00000007 /* R/W VME int VIRQ#7 for
MAILBOX0 */

/* sema0 - semaphore 0 register */

#define    SEMA0_SEM3         0x80000000 /* R/W semaphore 3          */

#define    SEMA0_SEM2         0x00800000 /* R/W semaphore 2          */

#define    SEMA0_SEM1         0x00008000 /* R/W semaphore 1          */

#define    SEMA0_SEM0         0x00000080 /* R/W semaphore 0          */

/* sema1 - semaphore 1 register */

#define    SEMA1_SEM7         0x80000000 /* R/W semaphore 7          */

#define    SEMA1_SEM6         0x00800000 /* R/W semaphore 6          */

#define    SEMA1_SEM5         0x00008000 /* R/W semaphore 5          */

#define    SEMA1_SEM4         0x00000080 /* R/W semaphore 4          */

/* mast_ctl - master control register */

#define    MAST_CTL_MRTRY_M  0xF0000000  /* Max PCI retries          */

#define    MAST_CTL_PWON_0   0x00000000  /* R/W posted write xfer count 128 */

#define    MAST_CTL_PWON_1   0x01000000  /* R/W posted write xfer count 256 */

#define    MAST_CTL_PWON_2   0x02000000  /* R/W posted write xfer count 512 */

#define    MAST_CTL_PWON_3   0x03000000  /* R/W posted write xfer count 1024
*/

#define    MAST_CTL_PWON_4   0x04000000  /* R/W posted write xfer count 2048
*/

#define    MAST_CTL_PWON_5   0x05000000  /* R/W posted write xfer count 4096
*/

#define    MAST_CTL_PWBBSY   0x0F000000  /* R/W posted write xfer count BUSY
*/
```

```
#define    MAST_CTL_VRL_0    0x00000000  /* R/W VMEbus request level 0      */
#define    MAST_CTL_VRL_1    0x00400000  /* R/W VMEbus request level 1      */
#define    MAST_CTL_VRL_2    0x00800000  /* R/W VMEbus request level 2      */
#define    MAST_CTL_VRL_3    0x00C00000  /* R/W VMEbus request level 3      */
#define    MAST_CTL_VRM_D    0x00000000  /* R/W VMEbus request mode
demand  */
#define    MAST_CTL_VRM_F    0x00200000  /* R/W VMEbus request mode fair    */
#define    MAST_CTL_VREL_R   0x00100000  /* R/W VMEbus request mode ROR
*/
#define    MAST_CTL_VREL_D   0x00000000  /* R/W VMEbus request mode RWD
*/
#define    MAST_CTL_VOWN_R   0x00000000  /* W VMEbus ownership release
*/
#define    MAST_CTL_VOWN_H   0x00080000  /* W VMEbus ownership hold       */
#define    MAST_CTL_VOWN_ACK 0x00040000  /* R VMEbus ownership due to
hold  */
#define    MAST_CTL_PABS_32  0x00000000  /* R/W PCI aligned burst size 32   */
#define    MAST_CTL_PABS_64  0x00001000  /* R/W PCI aligned burst size 64   */
#define    MAST_CTL_PABS_128 0x00002000  /* R/W PCI aligned burst size 128  */
#define    MAST_CTL_BUS_NO   0x000000FF  /* R/W PCI bus number MASK       */
/* misc_ctl - miscellaneous control register */
#define    MISC_CTL_VBTO_0   0x00000000  /* R/W VME bus time out disable    */
#define    MISC_CTL_VBTO_1   0x10000000  /* R/W VME bus time out 16 us      */
#define    MISC_CTL_VBTO_2   0x20000000  /* R/W VME bus time out 32 us      */
#define    MISC_CTL_VBTO_3   0x30000000  /* R/W VME bus time out 64 us      */
#define    MISC_CTL_VBTO_4   0x40000000  /* R/W VME bus time out 128 us     */
#define    MISC_CTL_VBTO_5   0x50000000  /* R/W VME bus time out 256 us     */
#define    MISC_CTL_VBTO_6   0x60000000  /* R/W VME bus time out 512 us     */
#define    MISC_CTL_VBTO_7   0x70000000  /* R/W VME bus time out 1024 us    */
#define    MISC_CTL_VARB_R   0x00000000  /* R/W VME arbitration Round Robin
*/
#define    MISC_CTL_VARB_P   0x04000000  /* R/W VME arbitration Priority    */
#define    MISC_CTL_VARBTO_1 0x00000000  /* R/W VME arb. time out disabled
*/
```

```
#define    MISC_CTL_VARBTO_2 0x01000000  /* R/W VME arb. time out 16 us     */
#define    MISC_CTL_VARBTO_3 0x02000000  /* R/W VME arb. time out 256 us    */
#define    MISC_CTL_SW_LRST  0x00800000  /* W software PCI reset            */
#define    MISC_CTL_SW_SRST  0x00400000  /* W software VME sysrest          */
#define    MISC_CTL_BI       0x00100000  /* R/W universe in BI-Mode         */
#define    MISC_CTL_ENGBI    0x00080000  /* R/W enable global BI initiator  */
#define    MISC_CTL_RESCIND  0x00040000  /* R/W enable rescinding DTACK     */
#define    MISC_CTL_SYSCON   0x00020000  /* R/W universe is sys controller  */
#define    MISC_CTL_V64AUTO  0x00010000  /* R/W initiate VME64 auto ID slave
*/
/* misc_stat - miscellaneous status register */
#define    MISC_STAT_ENDIAN     0x80000000  /* R always little endian mode  */
#define    MISC_STAT_LCLSIZE_32 0x00000000  /* R PCI bus size 32 bits       */
#define    MISC_STAT_LCLSIZE_64 0x40000000  /* R PCI bus size 64 bits       */
#define    MISC_STAT_DY4AUTO    0x08000000  /* R DY4 auto ID enable         */
#define    MISC_STAT_MYBBSY     0x00200000  /* R universe NOT busy          */
#define    MISC_STAT_DY4DONE    0x00080000  /* R DY4 auto ID done           */
#define    MISC_STAT_TXFE       0x00040000  /* R transmit FIFO empty        */
#define    MISC_STAT_RXFE       0x00020000  /* R receive FIFO empty         */
/* user_am - user AM codes register */
#define    USER_AM_1   0xFC000000  /* R/W user1 AM code MASK               */
#define    USER_AM_2   0x00FC0000  /* R/W user2 AM code MASK               */
/* vsi[x]_ctl - VMEbus slave image 0 control register */
#define    VSI_CTL_EN      0x80000000  /* R/W image enable                 */
#define    VSI_CTL_PWEN    0x40000000  /* R/W posted write enable          */
#define    VSI_CTL_PREN    0x20000000  /* R/W prefetch read enable         */
#define    VSI_CTL_AM_D    0x00400000  /* R/W AM code - data               */
#define    VSI_CTL_AM_P    0x00800000  /* R/W AM code - program            */
#define    VSI_CTL_AM_DP   0x00C00000  /* R/W AM code - both data & program
*/
#define    VSI_CTL_AM_U    0x00100000  /* R/W AM code - non priv           */
#define    VSI_CTL_AM_S    0x00200000  /* R/W AM code - supervisory        */
#define    VSI_CTL_AM_SU   0x00300000  /* R/W AM code - both user & super  */
```

```
#define   VSI_CTL_VAS_16  0x00000000  /* R/W address space A16          */
#define   VSI_CTL_VAS_24  0x00010000  /* R/W address space A24          */
#define   VSI_CTL_VAS_32  0x00020000  /* R/W address space A32          */
#define   VSI_CTL_VAS_R1  0x00030000  /* R/W address space reserved 1     */
#define   VSI_CTL_VAS_R2  0x00040000  /* R/W address space reserved 2     */
#define   VSI_CTL_VAS_R3  0x00050000  /* R/W address space reserved 3     */
#define   VSI_CTL_VAS_U1  0x00060000  /* R/W address space user 1       */
#define   VSI_CTL_VAS_U2  0x00070000  /* R/W address space user 2       */
#define   VSI_CTL_LD64EN  0x00000080  /* R/W enable 64 bit PCIbus xfers   */
#define   VSI_CTL_LLRMW   0x00000040  /* R/W enable PCIbus lock of VME RMW
*/
#define   VSI_CTL_LAS_M   0x00000000  /* R/W PCIbus memory space        */
#define   VSI_CTL_LAS_I   0x00000001  /* R/W PCIbus I/O space           */
#define   VSI_CTL_LAS_C   0x00000002  /* R/W PCIbus configuration space   */
/* vsi[x]_bs - VMEbus slave image 0 base address register */
#define   VSI0_BS     0xFFFFF000  /* R/W VME slave image 0 base add MASK  */
#define   VSI1_BS     0xFFFF0000  /* R/W VME slave image 1 base add MASK  */
#define   VSI2_BS     0xFFFF0000  /* R/W VME slave image 2 base add MASK  */
#define   VSI3_BS     0xFFFF0000  /* R/W VME slave image 3 base add MASK  */

/* vsi[x]_bd - VMEbus slave image 0 bound address register */
#define   VSI0_BD     0xFFFFF000  /* R/W VME slave image 0 bound add MASK */
#define   VSI1_BD     0xFFFF0000  /* R/W VME slave image 1 bound add MASK */
#define   VSI2_BD     0xFFFF0000  /* R/W VME slave image 2 bound add MASK */
#define   VSI3_BD     0xFFFF0000  /* R/W VME slave image 3 bound add MASK */
/* vsi[x]_to - VMEbus slave image 0 translation offset register */
#define   VSI0_TO     0xFFFFF000  /* R/W VME slave image 0 offset MASK    */
#define   VSI1_TO     0xFFFF0000  /* R/W VME slave image 1 offset MASK    */
#define   VSI2_TO     0xFFFF0000  /* R/W VME slave image 2 offset MASK    */
#define   VSI3_TO     0xFFFF0000  /* R/W VME slave image 3 offset MASK    */
/* lm_ctl - location monitor control */
#define   LM_CTL_EN       0x80000000  /* R/W location monitor enable      */
```

```
#define    LM_CTL_AM_D    0x00400000  /* R/W location monitor AM = DATA    */
#define    LM_CTL_AM_P    0x00800000  /* R/W location monitor AM = PROGRAM
*/
#define    LM_CTL_AM_DP   0x00C00000  /* R/W location monitor AM = BOTH
*/
#define    LM_CTL_AM_U    0x00100000  /* R/W location monitor AM = USER    */
#define    LM_CTL_AM_S    0x00200000  /* R/W location monitor AM = SUPER   */
#define    LM_CTL_AM_SU   0x00300000  /* R/W location monitor AM = BOTH    */
#define    LM_CTL_AM_16   0x00000000  /* R/W location monitor AM = A16     */
#define    LM_CTL_AM_24   0x00010000  /* R/W location monitor AM = A24     */
#define    LM_CTL_AM_32   0x00020000  /* R/W location monitor AM = A32     */
/* vrai_ctl - VMEbus register access image control register */
#define    VRAI_CTL_EN     0x80000000  /* R/W image enable             */
#define    VRAI_CTL_AM_D   0x00400000  /* R/W AM code - data           */
#define    VRAI_CTL_AM_P   0x00800000  /* R/W AM code - program        */
#define    VRAI_CTL_AM_DP  0x00C00000  /* R/W AM code - both           */
#define    VRAI_CTL_AM_U   0x00100000  /* R/W AM code - non priv       */
#define    VRAI_CTL_AM_S   0x00200000  /* R/W AM code - supervisory    */
#define    VRAI_CTL_AM_US  0x00300000  /* R/W AM code - both           */
#define    VRAI_CTL_VAS_16 0x00000000  /* R/W address space A16        */
#define    VRAI_CTL_VAS_24 0x00010000  /* R/W address space A24        */
#define    VRAI_CTL_VAS_32 0x00020000  /* R/W address space A32        */
/* vrai_bs - VMEbus register access image base address register */
#define    VRAI_BS   0xFFFFF000  /* R/W VME reg access image base add MASK */
/* vcsr_ctl - VMEbus CSR control register */
#define    VCSR_CTL_EN     0x80000000  /* R image enable               */
#define    VCSR_CTL_LAS_M  0x00000000  /* R/W PCIbus memory space      */
#define    VCSR_CTL_LAS_I  0x00000001  /* R/W PCIbus I/O space         */
#define    VCSR_CTL_LAS_C  0x00000002  /* R/W PCIbus configuration space  */
/* vcsr_to - VMEbus CSR translation offset */
#define    VCSR_TO   0xFFF80000  /* R/W VME CSR translation offset MASK   */
/* v_amerr - VMEbus AM code error log */
#define    V_AMERR_AMERR 0xFC000000   /* R AM codes for error log MASK   */
```

```
#define    V_AMERR_IACK   0x02000000    /* R VMEbus IACK                */
#define    V_AMERR_M_ERR  0x01000000    /* R multiple errors occurred   */
#define    V_AMERR_V_STAT 0x00800000    /* R/W VME error logs are valid */
/* vaerr - VMEbus address error log */
#define    VAERR      0xFFFFFFFF     /* R VMEbus address error log MASK */
/* vcsr_clr - VMEbus CSR bit clear register */
#define    VCSR_CLR_RESET   0x80000000 /* R/W board reset             */
#define    VCSR_CLR_SYSFAIL 0x40000000 /* R/W VMEbus sysfail          */
#define    VCSR_CLR_FAIL    0x20000000 /* R board fail                */
/* vcsr_set - VMEbus CSR bit set register */
#define    VCSR_SET_RESET   0x80000000 /* R/W board reset             */
#define    VCSR_SET_SYSFAIL 0x40000000 /* R/W VMEbus sysfail          */
#define    VCSR_SET_FAIL    0x20000000 /* R board fail                */
/* vcsr_bs - VMEbus CSR base address register */
#define    VCSR_BS    0xF8000000     /* R/W VME CSR base add MASK    */
```

Directory SRAM

The file in this directory can be used to test the integrity of the battery backed SRAM.

** FILE: TS.C

```
/*************************************************************************/
/* FILE: TS.C                                                            */
/*                                                                       */
/* Test battery backed SRAM with patterns and data=address.              */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
#include   <stdlib.h>
#include   <stdio.h>
#include   <dos.h>
unsigned char far * b_ptr;
unsigned int  far * w_ptr;
unsigned long far * l_ptr;
unsigned int  far * buf_ptr;
static unsigned long pat[4] = {
 0x55555555,
 0xCCCCCCCC,
 0x66666666,
 0xFFFFFFFF
};
void main( void ) {
unsigned long i, x;
unsigned char bdat;
unsigned char brd;
unsigned int  wdat;
unsigned int  wrd;
unsigned long ldat;
```

```
unsigned long lrd;
 printf("\nTesting 32K SRAM DATA/~DATA B/W/L/ADDR .....");
 buf_ptr = (unsigned int far *) MK_FP( 0xD800, 0x18 );
 /* fill and test buf with DATA/~DATA BYTES 4 patterns */
 for( x = 0; x < 4; x++ ) {
  b_ptr = (unsigned char far *) buf_ptr;
  bdat = (unsigned char) pat[x];
  for( i = 0x18; i < 0x8000; i++ ) {
   *b_ptr++ = bdat;
   bdat = ~bdat;
  }
  b_ptr = (unsigned char far *) buf_ptr;
  bdat = (unsigned char) pat[x];
  for( i = 0x18; i < 0x8000; i++ ) {
   brd = *b_ptr++;
   if( bdat != brd ) {
    printf("FAILED\nBYTE DATA @ ADDR: %Fp  WR: %.2X  RD: %.2X\n",
          --b_ptr, bdat, brd );
    exit( 1 );
   }
   bdat = (~bdat) & 0xFF;
  }
 }
 /* fill and test buf with DATA/~DATA WORDS 4 patterns */
 for( x = 0; x < 4; x++ ) {
  w_ptr = (unsigned int far *) buf_ptr;
  wdat = (unsigned int) pat[x];
  for( i = 0x18; i < 0x8000; i+=2 ) {
   *w_ptr++ = wdat;
   wdat = ~wdat;
  }
  w_ptr = (unsigned int far *) buf_ptr;
```

```
  wdat = (unsigned int) pat[x];
  for( i = 0x18; i < 0x8000; i+=2 ) {
    wrd = *w_ptr++;
    if( wdat != wrd ) {
      printf("FAILED\nWORD DATA @ ADDR: %Fp  WR: %.4X  RD: %.4X\n",
            --w_ptr, wdat, wrd );
      exit( 1 );
    }
    wdat = ~wdat;
  }
}
/* fill and test buf with DATA/~DATA LONGS 4 patterns */
for( x = 0; x < 4; x++ ) {
  l_ptr = (unsigned long far *) MK_FP( 0xD800, 0x18 ); ;
  ldat = (unsigned long) pat[x];
  for( i = 0x18; i < 0x8000; i+=4 ) {
    *l_ptr++ = ldat;
    ldat = ~ldat;
  }
  l_ptr = (unsigned long far *) MK_FP( 0xD800, 0x18 );
  ldat = (unsigned long) pat[x];
  for( i = 0x18; i < 0x8000; i+=4 ) {
    lrd = *l_ptr++;
    if( ldat != lrd ) {
      printf("FAILED\nLONG DATA @ ADDR: %Fp  WR: %.8X  RD: %.8X\n",
            --l_ptr, ldat, lrd );
      exit( 1 );
    }
    ldat = ~ldat;
  }
}
/* fill and test buf with DATA = ADD LONG */
```

```
  for( x = 0; x < 4; x++ ) {
   l_ptr = (unsigned long far *) MK_FP( 0xD800, 0x18 );
   for( i = 0x18; i < 0x8000; i+=4 ) {
    *l_ptr++ = i;
   }
   l_ptr = (unsigned long far *) MK_FP( 0xD800, 0x18 );
   for( i = 0x18; i < 0x8000; i+=4 ) {
    lrd = *l_ptr++;
    if( lrd != i ) {
     printf("FAILED\nDATA=ADDR @ ADDR: %Fp  WR: %.8X  RD: %.8X\n",
          --l_ptr, i, lrd );
     exit( 1 );
    }
   }
  }
  printf("PASSED\n\n");
  exit( 0 );
} /* end main */
```

Directory Timers

This directory contains sample code useful in the creation of applications involving the VMIVME-7698's three software controlled 16-bit timers. The code is written for the control of a single timer, but can be utilized in generating code for any timer configuration. The timers are described in Chapter 4 of the manual.

CPU.H

```
/*
** FILE: T7698T.H
**
*/
#define   DID_PWR_MGM    0x7113
#define   VID_PWR_MGM    0x8086
#define   IRQ5        0x0D
#define   GPI_T1       0x80   /* PIX General Purpose Input 15 (tmr 1) */
#define   GPI_T2       0x40   /* PIX General Purpose Input 14 (tmr 2) */
#define   GPI_T3       0x20   /* PIX General Purpose Input 13 (tmr 3) */
#define   GPO_T1       0x40   /* PIX General Purpose Output 30 (tmr 1)*/
#define   GPO_T2       0x10   /* PIX General Purpose Output 28 (tmr 2)*/
#define   GPO_T3       0x08   /* PIX General Purpose Output 27 (tmr 3)*/
#define   GPO_CLR      0xA7   /* PIX General Purpose Output CLR TMRS  */
#define   TIMER_CNTR1   0x00   /* Timer counter 1 offset          */
#define   TIMER_CNTR2   0x01   /* Timer counter 2 offset          */
#define   TIMER_CNTR3   0x02   /* Timer counter 3 offset          */
#define   TIMER_CNTL    0x03   /* Timer control offset           */
/*************************************************************************/
/* 8254 Control word                                                     */
/*************************************************************************/
#define   CW_SC0       0x00   /* W Selcect counter 0            */
#define   CW_SC1       0x40   /* W Selcect counter 1            */
#define   CW_SC2       0x80   /* W Selcect counter 2            */
#define   CW_RBC       0xC0   /* W Read back command            */
```

```
#define   CW_CLC       0x00   /* W Cntr latch command (cnt/stat)    */
#define   CW_SLC       0x00   /* W Status latch command             */
#define   CW_LSB       0x10   /* W LSB only                         */
#define   CW_MSB       0x20   /* W MSB only                         */
#define   CW_LSBMSB    0x30   /* W LSB first then MSB               */
#define   CW_M0        0x00   /* W Mode 0                           */
#define   CW_M1        0x02   /* W Mode 1                           */
#define   CW_M2        0x04   /* W Mode 2                           */
#define   CW_M3        0x06   /* W Mode 3                           */
#define   CW_M4        0x08   /* W Mode 4                           */
#define   CW_M5        0x0A   /* W Mode 5                           */
#define   CW_BCD       0x01   /* W Binary Coded Decimal             */
#define   CW_RB_CNT    0x00   /* W Read back count                  */
#define   CW_RB_STAT   0x00   /* W Read back status                 */
#define   CW_RB_C0     0x02   /* W Read back counter 0              */
#define   CW_RB_C1     0x04   /* W Read back counter 1              */
#define   CW_RB_C2     0x08   /* W Read back counter 2              */
```

** FILE: T_TIMERS.C

```
/*************************************************************************/
/* FILE:  T_TIMER.C                                                      */
/*                                                                       */
/*************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#include <dos.h>
#include "pci.h"
#include "cpu.h"
/* T_TIMER.C function prototypes */
void do_exit( int );
/* TIMERS.C function prototypes */
void far interrupt irq_rcvd( void );
void init_timer_int( void );
void restore_orig_int( void );
void load_counter( int, unsigned int );
void read_counter( int, unsigned int *, unsigned char * );
/* global variables */
unsigned char bus, dev_func;
/* the following globals are used in other files as 'extern' variables */
unsigned char tmr_status, t1_stat, t2_stat, t3_stat;
unsigned int tmr_cnt, t1_cnt, t2_cnt, t3_cnt;
unsigned long t1_count;     /* counts no. of times timer 1 ISR entered */
unsigned long t2_count;     /* counts no. of times timer 2 ISR entered */
unsigned long t3_count;     /* counts no. of times timer 3 ISR entered */
unsigned int pwr_mgm_base;
unsigned int gpi_base;
unsigned int gpo_base;
```

```
unsigned int timer_base;
unsigned char pic1_org;
unsigned char gpo_org;
void main( int argc, char * argv[])
{
 unsigned long t1, t2, t3;
 int test_int;
 unsigned long temp_dword;
 char user[80];
 timer_base = 0x500;
 /* try to locate the power management device on the PCI bus */
 test_int = find_pci_device(DID_PWR_MGM, VID_PWR_MGM, 0,
          &bus, &dev_func);
 if(test_int != SUCCESSFUL)
 {
   printf("\nUnable to locate power management device on PCI bus\n");
   do_exit( 1 );
 }
 /* get base address from config area */
 test_int = read_configuration_area(READ_CONFIG_DWORD,
                    bus, dev_func, 0x40, &temp_dword);
 if(test_int != SUCCESSFUL)
 {
   printf("\nUnable to read POWER MGM. BASE ADDRESS @ 0x40 in config
space\n");
   do_exit( 1 );
 }
 pwr_mgm_base = temp_dword & 0x0000FFC0;
 gpi_base = pwr_mgm_base + 0x31; /* PIX general purpose input bits 8-15 */
 gpo_base = pwr_mgm_base + 0x37; /* PIX general purpose output bits 24-31 */
 disable();
 /* Read 8259 slave Programmable Interrupt controller */
```

```
pic1_org = inp(0x21) & 0xFF; /* slave mask bits */
/* disable interrupt 5 */
outp(0x21, (pic1_org | 0x20) ); /* 0 = enable 1 = disable */
enable();
gpo_org = inp( gpo_base ) & 0xFF;
/* setup timers interrupt service routine */
init_timer_int();
/*
** verify all three counters can generate an interrupt (counters 1,2,3)
*/
printf("\nTesting all three 16 bit counters for interrupt ....");
/* setup for interrupts to occur */
t1_count = 0;
t2_count = 0;
t3_count = 0;
t1 = 0;
t2 = 0;
t3 = 0;
tmr_status = 0;
test_int = 100;
/* load counters */
load_counter( 1, 0xFFFF );
do
{
 if( t1_count ) {
   t1++;
   break;
 }
 test_int--;
 delay( 1 );
} while( test_int );
/* disable timers by reloading the control word */
```

```
outp( timer_base + TIMER_CNTL, (CW_SC0 | CW_LSBMSB | CW_M2) );
tmr_status = 0;
test_int = 100;
load_counter( 2, 0xFFFF );
do
{
 if( t2_count ) {
   t2++;
   break;
 }
 test_int--;
 delay( 1 );
} while( test_int );
/* disable timers by reloading the control word */
outp( timer_base + TIMER_CNTL, (CW_SC1 | CW_LSBMSB | CW_M2) );
tmr_status = 0;
test_int = 100;
load_counter( 3, 0xFFFF );
do
{
 if( t3_count ) {
   t3++;
   break;
 }
 test_int--;
 delay( 1 );
} while( test_int );
/* disable timers by reloading the control word */
outp( timer_base + TIMER_CNTL, (CW_SC2 | CW_LSBMSB | CW_M2) );
/* clear all three status bits in GPI */
outp( gpo_base, ( gpo_org & GPO_CLR ) );
/* set all three GPO outputs to 1 to allow int status registers to function */
```

```
  outp( gpo_base, ( gpo_org | GPO_T1 | GPO_T2 | GPO_T3 ) );
  if( t1 && t2 && t3 )
  {
    printf("PASSED\n");
  }
  else
  {
    printf("FAILED\n");
    if( !t1 ) printf("TIMER 1 failed\n");
    if( !t2 ) printf("TIMER 2 failed\n");
    if( !t3 ) printf("TIMER 3 failed\n");
    do_exit( 2 );
  }
  /* do orderly exit */
  do_exit( 3 );
} /* end main */
void do_exit( int xit_code )
{
  if( xit_code > 1 )  restore_orig_int();
  outp( gpo_base, gpo_org );
  if( xit_code == 3 ) xit_code = 0;
  exit( xit_code );
} /* do_exit */
** FILE: TIMERS.C
/*
** FILE: TIMERS.C
**
*/
#include <stdlib.h>
#include <stdio.h>
#include <dos.h>
#include <ctype.h>
```

```
#include <conio.h>
#include "cpu.h"
/* function prototypes */
void far interrupt irq_rcvd( void );
void init_timer_int( void );
void restore_orig_int( void );
void load_counter( int, unsigned int );
void read_counter( int, unsigned int *, unsigned char * );
/* global variables */
extern unsigned long t1_count;    /* timer 1 count */
extern unsigned long t2_count;    /* timer 2 count */
extern unsigned long t3_count;    /* timer 3 count */
extern unsigned char tmr_status;
extern unsigned int gpi_base;
extern unsigned int gpo_base;
extern unsigned int timer_base;
extern unsigned char pic1_org;
extern unsigned char gpo_org;
void far interrupt (* old_vect)(void);
/*****************************************************************/
/* init_timer_int()                                              */
/*                                                               */
/* purpose: Using the interrupt assigned, the original vector is */
/*        saved and the vector to the new ISR is installed. The  */
/*        programmable-interrupt-controller (PIC) is enabled.    */
/*                                                               */
/* Prerequisite: The interrupt line to be used must have         */
/*           already been loaded in the global variable.         */
/*                                                               */
/*****************************************************************/
/* parameters: none                                              */
/*****************************************************************/
```

```
/*  return value: none                                  */
/*********************************************************************/
void init_timer_int( void )
{
 disable();
 old_vect = getvect( IRQ5 ); /* save vector for IRQ5 */
 setvect( IRQ5, irq_rcvd );
 /* enable interrupt 5 */
 outp(0x21, (pic1_org & 0xDF) ); /* 0 = enable 1 = disable */
 /* clear all three GPO inputs */
 outp( gpo_base, ( gpo_org & GPO_CLR ) );
 /* set all three GPO outputs to 1 to allow int status registers to function */
 outp( gpo_base, (gpo_org | GPO_T1 | GPO_T2 | GPO_T3 ) );
 enable();
} /* init_timer_int */
/*********************************************************************/
/*  restore_orig_int()                                 */
/*                                                     */
/*  purpose: Using the interrupt assigned, the original vector is  */
/*        restored and the programmable-interrupt-controller    */
/*        is disabled.                                 */
/*                                                     */
/*  Prerequisite: The interrupt line to be used must have        */
/*           already been loaded in the global variable.     */
/*                                                     */
/*********************************************************************/
/*  parameters: none                                   */
/*********************************************************************/
/*  return value: none                                 */
/*********************************************************************/
void restore_orig_int( void )
{
```

```
 disable();
 outp(0x21, pic1_org);
 setvect( IRQ5, old_vect );
 enable();
} /* restore_orig_int */
/*****************************************************************/
/*  load_counter()                                               */
/*                                                               */
/*  purpose: Loads the appropriate counter with the count passed */
/*                                                               */
/*                                                               */
/*****************************************************************/
/*  parameters: int counter = 1, 2, 3 for COUNTER 1, 2, or 3     */
/*              unsigned int count = count to be loaded          */
/*****************************************************************/
/*  return value: none                                           */
/*****************************************************************/
void load_counter( int counter, unsigned int count )
{
 int lsb, msb;
 lsb = count & 0xff;
 msb = count >> 8;
 switch( counter )
 {
  case 1: /* select counter 1, LSB then MSB, mode 2 */
    outp( timer_base + TIMER_CNTL, (CW_SC0 | CW_LSBMSB | CW_M2) );
    outp( timer_base + TIMER_CNTR1, (unsigned char) lsb );
    outp( timer_base + TIMER_CNTR1, (unsigned char) msb );
  break;
  case 2: /* select counter 2, LSB then MSB, mode 2 */
    outp( timer_base + TIMER_CNTL, (CW_SC1 | CW_LSBMSB | CW_M2) );
    outp( timer_base + TIMER_CNTR2, (unsigned char) lsb );
```

```
    outp( timer_base + TIMER_CNTR2, (unsigned char) msb );
   break;
   case 3: /* select counter 3, LSB then MSB, mode 2 */
    outp( timer_base + TIMER_CNTL, (CW_SC2 | CW_LSBMSB | CW_M2) );
    outp( timer_base + TIMER_CNTR3, (unsigned char) lsb );
    outp( timer_base + TIMER_CNTR3, (unsigned char) msb );
   break;
  }
} /* load_counter */
/*********************************************************************/
/*  read_counter()                                                   */
/*                                                                   */
/*  purpose: Reads the appropriate counter in the appropriate        */
/*           bank with the remaining count and status.               */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/*  parameters: int counter = 1, 2, 3 for COUNTER 1, 2, or 3         */
/*              unsigned int * count = remaining count               */
/*              unsigned char * status = counter status              */
/*********************************************************************/
/*  return value: none                                               */
/*********************************************************************/
void read_counter( int counter,
                   unsigned int * count, unsigned char * status )
{
 int lsb, msb;
 switch( counter )
 {
  case 1: /* select counter 1, LSB then MSB */
   outp( timer_base + TIMER_CNTL, ( CW_RBC | CW_RB_CNT | CW_RB_STAT |
CW_RB_C0 ) );
```

```
    *status = inp( timer_base + TIMER_CNTR1 ) & 0xFF;
    lsb = inp( timer_base + TIMER_CNTR1 ) & 0xFF;
    msb = inp( timer_base + TIMER_CNTR1 ) & 0xFF;
    msb = msb << 8;
    *count = ( lsb | msb );
  break;
  case 2: /* select counter 2, LSB then MSB */
    outp( timer_base + TIMER_CNTL, ( CW_RBC | CW_RB_CNT | CW_RB_STAT |
CW_RB_C1 ) );
    *status = inp( timer_base + TIMER_CNTR2 ) & 0xFF;
    lsb = inp( timer_base + TIMER_CNTR2 ) & 0xFF;
    msb = inp( timer_base + TIMER_CNTR2 ) & 0xFF;
    msb = msb << 8;
    *count = ( lsb | msb );
  break;
  case 3: /* select counter 3, LSB then MSB */
    outp( timer_base + TIMER_CNTL, ( CW_RBC | CW_RB_CNT | CW_RB_STAT |
CW_RB_C2 ) );
    *status = inp( timer_base + TIMER_CNTR3 ) & 0xFF;
    lsb = inp( timer_base + TIMER_CNTR3 ) & 0xFF;
    msb = inp( timer_base + TIMER_CNTR3 ) & 0xFF;
    msb = msb << 8;
    *count = ( lsb | msb );
  break;
 }
} /* read_counter */
/*****************************************************************/
/*  irq_rcvd()                                                   */
/*                                                               */
/*  purpose: Interrupt service routine used to service any of the */
/*        counters on the 7698.                                  */
/*                                                               */
/*                                                               */
```

```
/*********************************************************************/
/*  parameters: none                                                 */
/*********************************************************************/
/*  return value: none                                               */
/*********************************************************************/
void interrupt irq_rcvd(void)
{
 disable();
 asm {
  .386P
  push  eax
  push  ebx
 }
 tmr_status = inp( gpi_base ) & 0xFF;
 /* increment counts and clear status */
 if( tmr_status & GPI_T1 ) {
  t1_count++;
  outp( gpo_base, (gpo_org & (~GPO_T1)) );  /* clear timer 1 status bit */
 }
 if( tmr_status & GPI_T2 ) {
  t2_count++;
  outp( gpo_base, (gpo_org & (~GPO_T2)) );  /* clear timer 2 status bit */
 }
 if( tmr_status & GPI_T3 ) {
  t3_count++;
  outp( gpo_base, (gpo_org & (~GPO_T3)) );  /* clear timer 3 status bit */
 }
 outp( gpo_base, (gpo_org | GPO_T1 | GPO_T2 | GPO_T3) ); /* enable status */
 /* Non specific end of interrupt to PIC */
 outp(0x20, 0x20);  /* Master end of irq command */
 asm {
  .386P
```

```
    pop ebx
    pop eax
  }
  enable();
}
```

Directory WATCHDOG

This directory contains sample code useful in the creation of applications involving the VMIVME-7698's watchdog timer function as described in Chapter 4.

** FILE: WATCHDOG.H

```
/*
** DS1384 REGISTER OFFSETS
*/
                        /*7 6 5 4 3 2 1 0*/
#define     CLK_MSEC     0x00   /* 00-99                  */
#define     CLK_SEC      0x01   /* 00-59 0                */
#define     CLK_MIN      0x02   /* 00-59 0                */
#define     CLK_MINAL    0x03   /* 00-59 M                */
#define     CLK_HRS      0x04   /* 01-12+A/P OR 00-23     */
#define     CLK_HRSAL    0x05   /* 01-12+A/P OR 00-23     */
#define     CLK_DAY      0x06   /* 01-07 0 0 0 0 0        */
#define     CLK_DAYAL    0x07   /* 01-07 M 0 0 0 0        */
#define     CLK_DATE     0x08   /* 01-31 0 0              */
#define     CLK_MONTH    0x09   /* 01-12    0             */
#define     CLK_YRS      0x0A   /* 00-99                  */
#define     WD_CMD       0x0B   /* command register       */
#define     WD_MSEC      0x0C   /* milli second watchdog time  */
#define     WD_SEC       0x0D   /* seconds watchdog time  */
/*
** DS1384 COMMAND REGSITER BIT DEFINITIONS
*/
#define     WD_TE        0x80   /* transfer enable 1 - allow updates */
#define     WD_IPSW      0x40   /* interrupt switch 0 - WD out INTA  */
#define     WD_IBHL      0x20   /* int. B output 0 - current sink    */
#define     WD_PU        0x10   /* pulse/level 1 - 3 ms pulse        */
#define     WD_WAM       0x08   /* watchdog alarm mask 0 - active    */
#define     WD_TDM       0x04   /* time-of-day alarm mask 0 - active */
#define     WD_WAF       0x02   /* watchdog alarm flag    */
#define     WD_TDF       0x01   /* time-of-day flag       */
```

** FILE:WDTO.C

```
/*************************************************************************/
/* FILE: WDTO_RST.C                                                      */
/*                                                                       */
/* Setup watchdog to issue reset on time out.                            */
/*                                                                       */
/*                                                                       */
/*************************************************************************/
#include   <stdlib.h>
#include   <stdio.h>
#include   <dos.h>
#include   <time.h>
#include   <conio.h>
#include   <ctype.h>
#include   "watchdog.h"
unsigned char far * wd_ptr;
time_t t;
char usr[80];
char reg_b;
void main( void ) {
  /* Install WATCHDOG RESET Jumper */
  wd_ptr = (unsigned char far *) MK_FP( 0xD800, 0 );
  /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  *(wd_ptr + WD_CMD) = ( WD_TE | WD_WAM );
  *(wd_ptr + WD_MSEC) = 0; /* load with 0 to disable */
  *(wd_ptr + WD_SEC) = 0;  /* load with 0 to disable */
  *(wd_ptr + WD_CMD) = ( WD_TE | WD_WAM ); /* allow update with 0 time */
  *(wd_ptr + WD_CMD) = WD_WAM; /* set watchdog alarm mask to 1 */
  *(wd_ptr + WD_MSEC) = 0x99; /* 00.99 seconds */
  *(wd_ptr + WD_SEC) = 0x99;  /* 99.00 seconds */
  *(wd_ptr + WD_CMD) = ( WD_TE | WD_PU ); /* set for 3 ms pulse */
  printf("Reset time out in 99.99 seconds\n\n");
```

```
  time(&t);
  printf("START DATE & TIME: %.24s\n\n", ctime(&t) );
  do {
    time(&t);
    printf("CURRENT DATE & TIME: %.24s\r", ctime(&t) );
    delay( 250 );
  } while( !kbhit() );
} /* end main */
```

Index

J

K

L

M

N

O

P

R

S

T

U

V