VMIVME-7696

Pentium II® Processor-Based VMEbus CPU

Product Manual

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

500-007696-000 Rev. A
22-Jan-1999

VMIC

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

COPYRIGHT AND TRADEMARKS

© Copyright 1999. The information in this document has been carefully checked and is believed to be entirely reliable. While all reasonable efforts to ensure accuracy have been taken in the preparation of this manual, VMIC assumes no responsibility resulting from omissions or errors in this manual, or from the use of information contained herein.

VMIC reserves the right to make any changes, without notice, to this or any of VMIC’s products to improve reliability, performance, function, or design.

VMIC does not assume any liability arising out of the application or use of any product or circuit described herein; nor does VMIC convey any license under its patent rights or the rights of others.

For warranty and repair policies, refer to VMIC’s Standard Conditions of Sale.

AMXbus, BITMODULE, COSMODULE, DMAbus, Instant OPC wizard logo, IOWorks Access, IOWorks Foundation, IOWorks man figure, IOWorks Manager, IOWorks Server, MAGICWARE, MEGAMODULE, PLC ACCELERATOR (ACCELERATION), Quick Link, RTnet, Soft Logic Link, SRTbus, TESTCAL, “The Next Generation PLC”, The PLC Connection, TURBOMODULE, UCLIO, UIOD, UPLC, Visual Soft Logic Control(ler), *VMEaccess*, *VMEmanager*, *VMEmonitor*, VMEnet, VMEnet II, and *VMEprobe* are trademarks. The I/O Experts, The I/O Systems Experts, The Soft Logic Experts, and The Total Solutions Provider are service marks of VMIC.

(I/O man figure)

(Instant OPC wizard logo)

(IOWorks man figure)

The I/O man figure, IOWorks, UIOC, Visual IOWorks, and *WinUIOC* are registered trademarks of VMIC.

ActiveX is a trademark and Microsoft, Microsoft Access, MS-DOS, Visual Basic, Visual C++, Win32, Windows, Windows NT, and XENIX are registered trademarks of Microsoft Corporation.

MMX is a trademark and Pentium is a registered trademark of Intel Corporation.

PICMG and CompactPCI are registered trademarks of PCI Industrial Computer Manufacturers’ Group.

Other registered trademarks are the property of their respective owners.

VMIC

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

Table of Contents

List of Figures

List of Tables

Overview

Introduction

VMIC's VMIVME-7696 is a complete IBM PC/AT-compatible Pentium II processor-based computer with the additional benefits of Dual Eurocard construction and full compatibility with the VMEbus Specification Rev. C.1. The VMIVME-7696 with advanced VMEbus interface and RAM that is dual-ported to the VMEbus, is ideal for multiprocessor applications.

The dual-slot CPU board functions as a standard PC/AT, executing a PC/AT-type power-on self-test, then boots up MS-DOS, Windows 3.11, Windows 95, Windows NT, or any other PC/AT-compatible operating system. The PC/AT mode of the VMIVME-7696 is discussed in Chapter 3 of this manual.

The VMIVME-7696 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-7696 may be accessed as a VMEbus slave board. The VMEbus functions are available by programming the VMIVME-7696'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-7696 Tundra Universe II™-Based VMEbus Interface Product Manual (document No. 500-007696-001 Rev. A).*

The VMIVME-7696 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-7696 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, remote LANboot, 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-7696 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-7696.

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 for Workgroups Version 3.11, Windows 95, or Windows NT (Versions 3.5x and 4.0) operating systems.

Appendix C- Basic Input/Output System describes the menus and options associated with the Award (system) BIOS.

Appendix D - LANWorks BIOS describes the menus and options associated with the LANWorks BIOS.

Appendix E - SCSI Selection Utility describes the menus and options associated with the Adaptec SCSI BIOS.

Appendix F - 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 G - 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-7696, please refer to:

VMIC specification number 800-007696-000

The following books refer to the Tundra Universe II-based interface option available in the VMIVME-7696:

VMIVME-7696, Tundra Universe II™-Based VMEbus Interface Product Manual
VMIC Doc. No. 500-007696-001

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-7696 include:

Intel Pentium II Processor at 233 MHz, 266 MHz, 300 MHz, and 333 Mhz
January 1998, Order Number: 243335-003
Intel Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
(800) 548-4752
www.intel.com

Intel Pentium II Processor at 350 MHz, 400 MHz, and 450 MHz
Intel Literature Sales
P.O. Box 7641
Mt. Prospect, IL 60056-7641
(800) 548-4752
www.intel.com

Intel 440BX AGP set: 82443BX Host Bridge/Controller
April 1998, Order Number: 290633-001
Intel Corporation
P.O. Box 58119
Santa Clara, CA 95052-8119
(408) 765-8080
www.intel.com

Intel 82371EB PCI-to-ISA/IDE Xcelerator (PIIX4E)

April 1997, Order Number:290562-001
Intel Corporation
P.O. Box 58119
Santa Clara, CA 95052-8119
(408) 765-8080
www.intel.com

Award BIOS

Award Software International, Inc.
777 East Middle Field Road
Mountain View, CA 94043-4023
(650) 237-6800 FAX: (650) 968-0274 BBS: (650) 968-0249
www.award.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. 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-7696:

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.

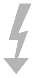
Dangerous Procedure Warnings

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

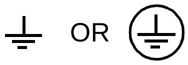
WARNING

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

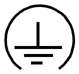
Safety Symbols Used in This Manual



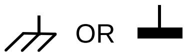
Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 V are so marked).



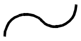
Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating equipment.



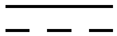
Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. Before operating the equipment, terminal marked with this symbol must be connected to ground in the manner described in the installation (operation)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



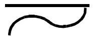
Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.



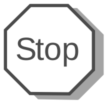
Alternating current (power line).



Direct current (power line).

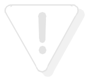


Alternating or direct current (power line).



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

The WARNING sign 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.



The CAUTION sign 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.

The NOTE sign 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-7696 Features and Options

Contents

Introduction

The VMIVME-7696 performs all the functions of a standard IBM PC/AT motherboard with the following features:

- Dual-slot VMEbus 6U size
- Includes a high-performance Intel Pentium II processor
- Low power, split voltage-based design
- Up to 256 Mbyte of Synchronous DRAM
- 64-bit AGP SVGA video graphics accelerator
  - 4 Mbyte SGRAM Video Memory
  - Resolutions up to 1,600x1200x64K colors
- Battery-backed clock/calendar
- Front panel reset switch and miniature speaker
- On-board ports for a keyboard and mouse, Ultra-IDE hard drive, floppy drive, SCSI, Ethernet, video, dual serial, and parallel I/O
- Front panel “vital sign” indicators (power, Ultra-IDE hard drive activity, VMEbus SYSFAIL, and Ethernet status)
- Three general-purpose programmable 16/32-bit timers
- Software-controlled watchdog timer
- Up to 48 Mbyte of bootable flash on secondary IDE
- 128 Kbyte of battery-backed SRAM

The VMIVME-7696 supports standard PC/AT I/O features such as those listed in Table 1-1. Figure 1-1 on page 27 shows a block diagram of the VMIVME-7696 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 Submini-D 9-Pin
One Enhanced Parallel Port, Supports ECP/EPP Modes	Parallel	Front Panel Submini-D 25-Pin
AT-Style Keyboard Controller with a PS/2-Style Adapter	Keyboard	Front Panel PS/2-Style Connector, Mini-DIN Circular (female)
AT-Style Mouse Controller with a PS/2-Style Adapter	Mouse	Front Panel PS/2-Style Connector, Mini-DIN Circular (female)
AGP Video Controller with 4 Mbyte DRAM	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 (Bottom Card)
Ultra IDE Fixed Disk Controller (two drives maximum)	Drives C, D	P2 (Bottom Card)
Ultra/Fast/Wide SCSI II	SCSI	P2 (Top Card)
Universal Serial Bus	USB	Front Panel USB Connector
Hardware Reset	RST	Front Panel Push-Button
IBM/PC Sound		Front Panel Speaker Port
Power Status, Hard Drive Activity, and Ethernet Status	LED Indicators	Front Panel

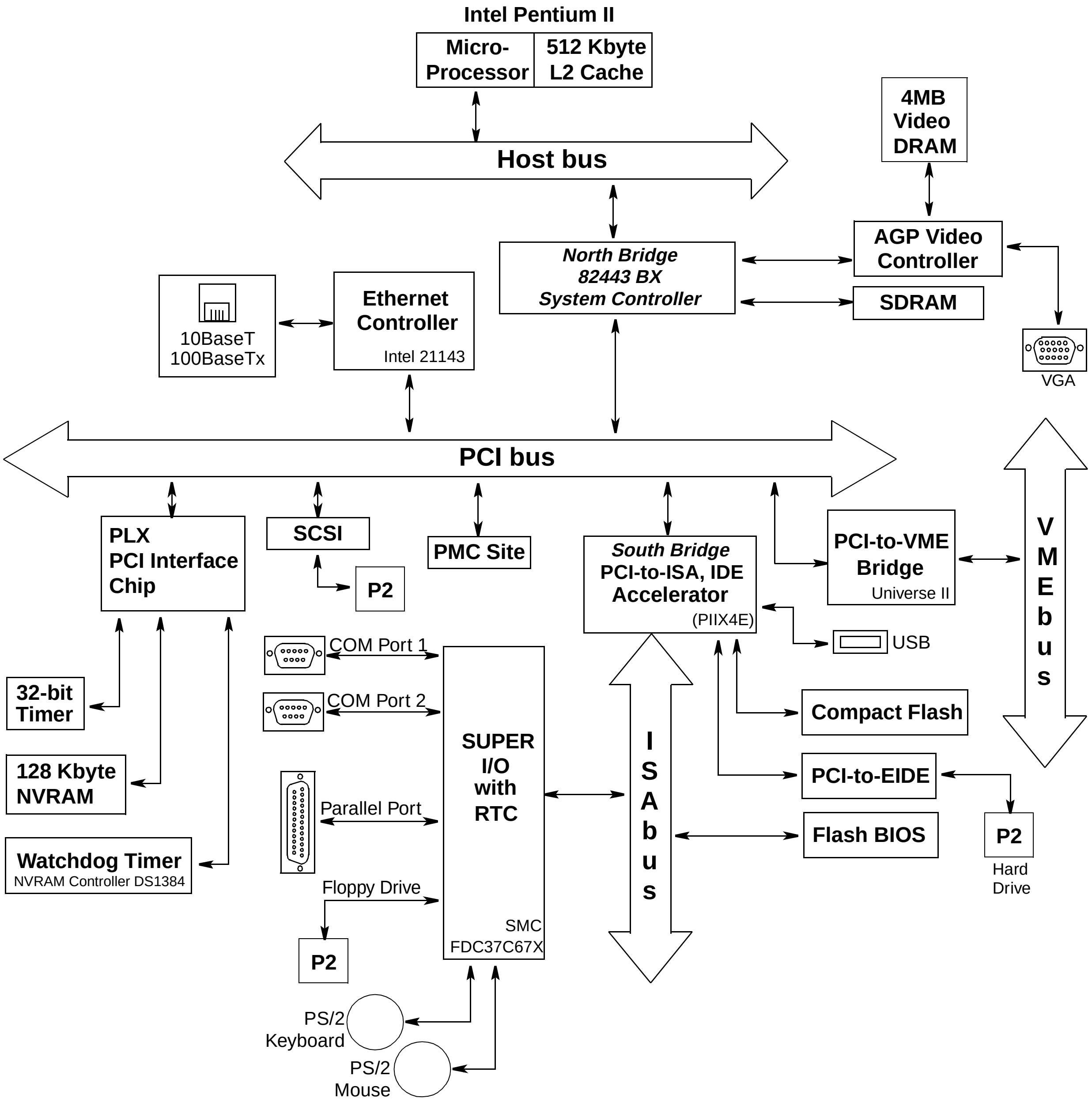


Figure 1-1 VMIVME-7696 Block Diagram

VMEbus Features

In addition to its PC / AT functions, the VMIVME-7696 has the following VMEbus features:

- Dual-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-7696 functions in a typical VMEbus system. The VMIVME-7696 is a versatile dual-board solution for VMEbus control with familiar PC / AT operation.

The VMIVME-7696 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-7696 Tundra Universe II Based VMEbus Interface Product Manual*.

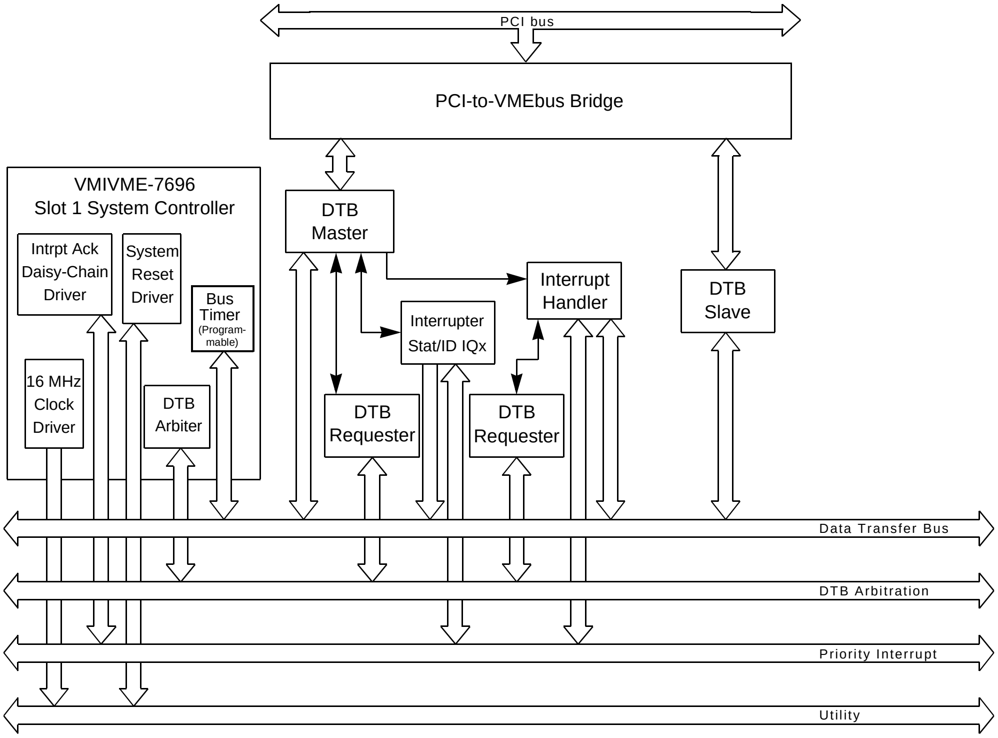


Figure 1-2 VMIVME-7696 VMEbus Functions

VMIVME-7696 Product Options

VMIC’s VMIVME-7696 is built around three fundamental hardware configurations. These involve processor performance, SDRAM memory size, and Compact Flash 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-7696 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-7696. 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).

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-7696 is factory populated with user-specified options as part of the VMIVME-7696 ordering information. The CPU speed and SDRAM size are not user-upgradable. To change CPU speeds or RAM size, contact customer service to receive a Return Material Authorization (RMA).

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

The VMIVME-7696 is tested for system operation and shipped with factory-installed header jumpers. The physical location of the jumpers and connectors for the dual board CPU are illustrated in Figure 2-1 on page 33 and Figure 2-4 on page 42. The definitions of the CPU board jumpers and connectors are included in Table 2-1 through Table 2-16. The Tundra Universe II-based PCI-to-VMEbus Bridge jumper configuration is discussed in a following section.

All jumpers are factory configured and should not be modified by the user. There are six exceptions: Password Clear (E3), Watchdog Timer (E18), VMEbus System Reset Driver (E6), VMEbus System Reset Receiver (E7), VMEbus SYSFAIL On Reset (E8), and the NVRAM Battery Connection (E4).

Modifying any other jumper will void the Warranty and may damage the unit. The default jumper condition of the VMIVME-7696 is expressed in Table 2-1 through Table 2-16 with **bold text** in the table cells.

In order to gain access to these five jumpers (on Rev A boards) the user may have to remove the VMIVME-7696 front panel. If required please follow the steps below to remove the front panel.

1. Unplug COM1 and COM2 cables from Headers E2 and E14.
2. Remove the jack screws from the parallel port and SVGA port.
3. Remove the 2 front panel screws from the bottom of the front panel (see Figure 2-1).
4. Remove the screw from the top of the front panel.
5. Carefully pull the front panel away from the unit.

When the jumper modifications are complete, reattach the front panel. Ensure that all the front panel screws and jack screws are reinstalled.

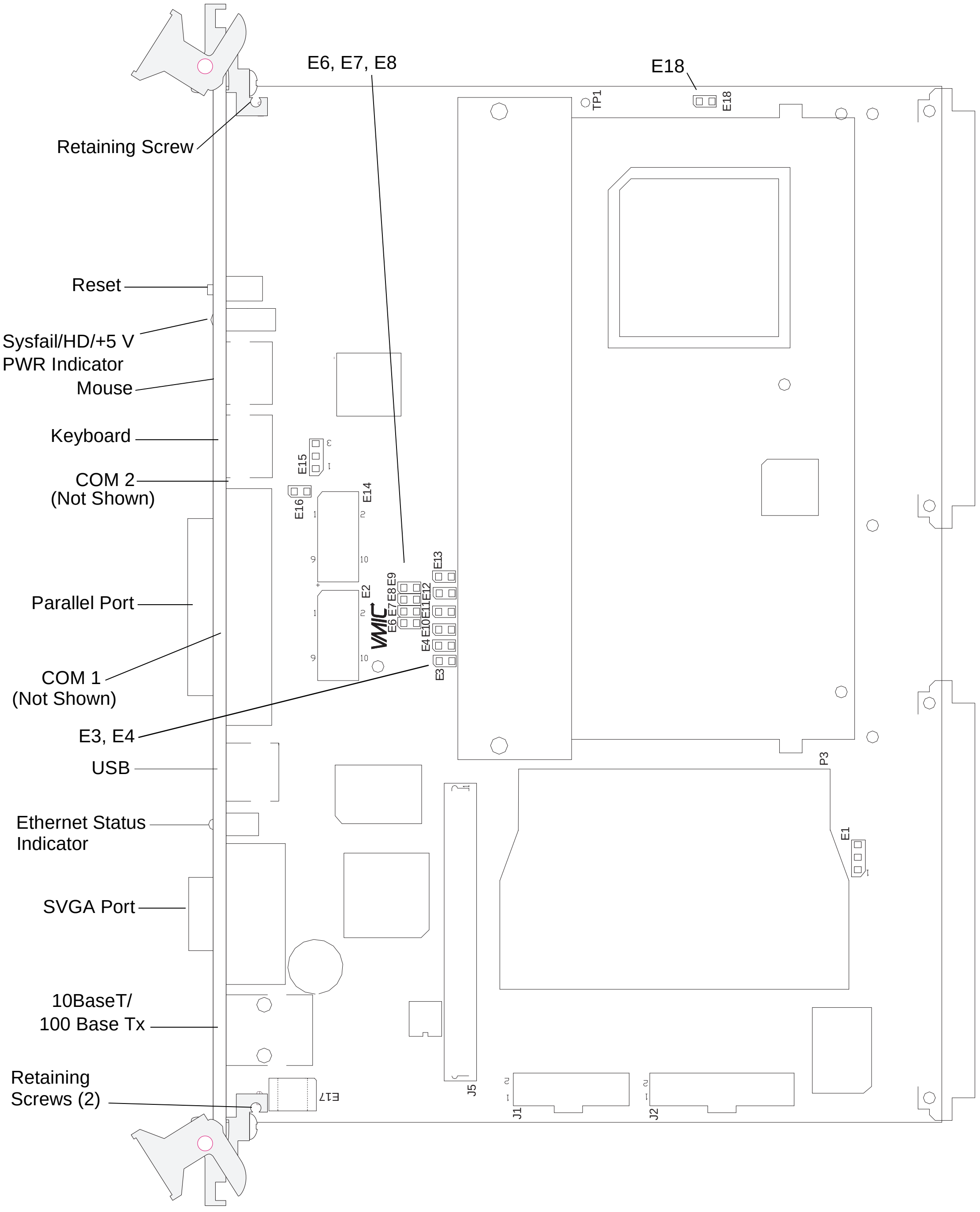


Figure 2-1 VMIVME-7696 CPU Board, I/O Port, and Jumper Locations (Rev A)

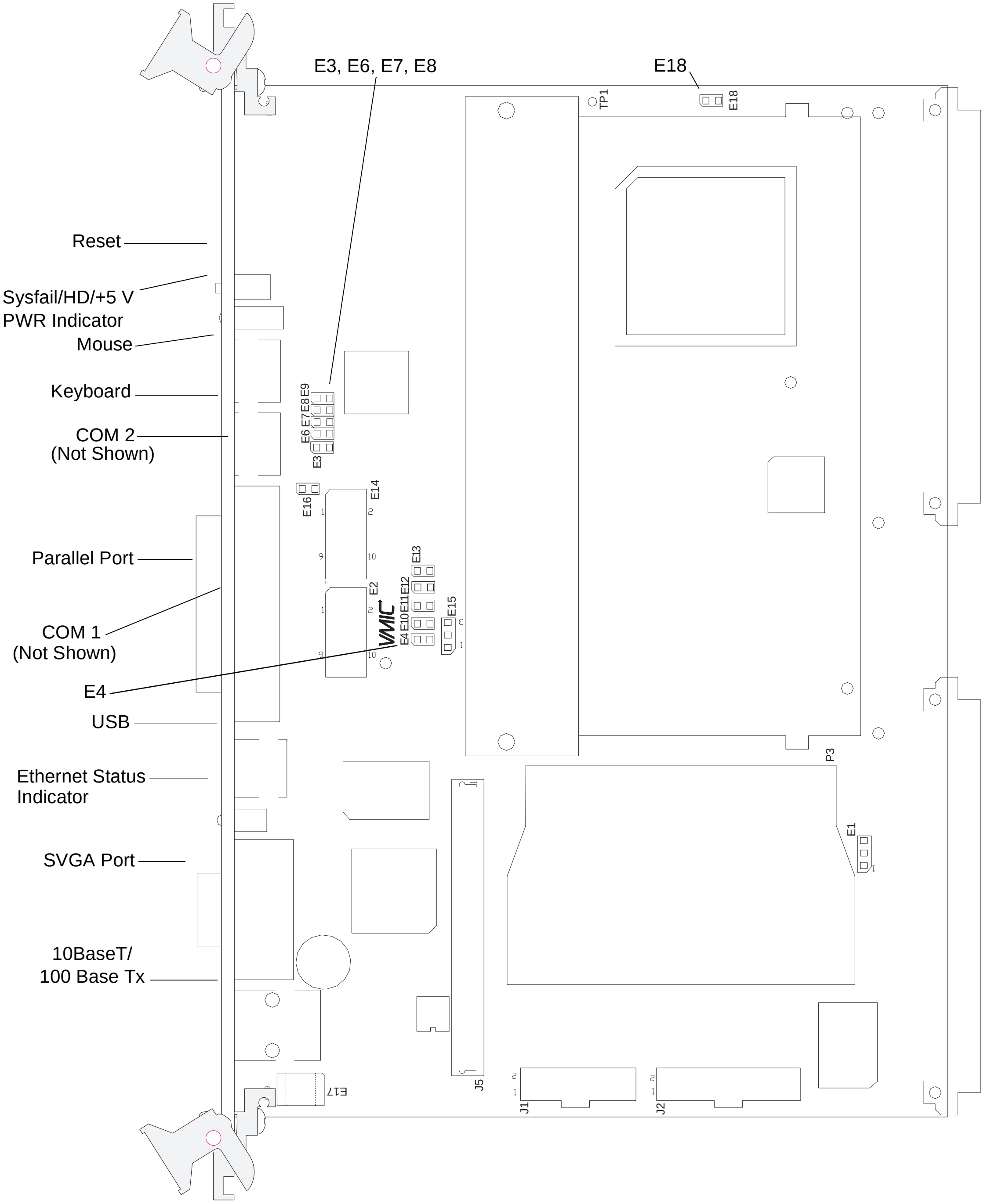


Figure 2-2 VMIVME-7696 CPU Board, I/O Port, and Jumper Locations (Rev B)

Table 2-1 VMIVME-7696C Board Connectors

Connector	Function
J1	ITP Connectors
J2	Port 80 Connector
J3	Ethernet Connector
J4	Video Connector
J5	Board to Board Connector
J6	PS/2 Keyboard Connector
J7	PS/2 Mouse Connector
P1	VME Connector
P2	VME Connector
P3	SO DIMM Connector
P4	Parallel Port Connector
P5	USB Connector
E2	Serial Port Header
E14	Serial Port Header
E17	EPLD Programming Header

Table 2-2 Boot Block Lock - Jumper (E1)

Select	Pins
Boot Block Unlocked	1 and 2
Boot Block Locked	**2 and 3**

Note

The VMIVME-7696’s BIOS has the capability (Default: Disabled) of password protecting casual access to the unit’s CMOS set-up screens. The Password Clear jumper (E33) 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 E33
3. Power up the unit
4. Turn off the power to the unit and remove the jumper from E33

When power is reapplied to the unit, the CMOS will be cleared.

Table 2-3 Clear CMOS - Jumper (E3)

Select	Jumper Position
Clear CMOS	In
Retain CMOS	**Out**

Table 2-4 Bus to Core Frequency - Jumpers (E4, E10, E11, E12)

Processor Frequency Bus Frequency	333MHz 66MHz	450MHz 100MHz
Jumper E4	In	In
Jumper E10	In	Out
Jumper E11	Out	In
Jumper E12	Out	Out

Table 2-5 VME Bus System Reset Driver - Jumper (E6)

Select	Jumper Position
Active	**In**
Disabled	**Out**

Table 2-6 VME Bus System Reset Receiver - Jumper (E7)

Select	Jumper Position
Active	**In**
Disabled	Out

Table 2-7 VME Bus SYSFAIL On Reset - Jumper (E8)

Select	Jumper Position
Active	**In**
Disabled	Out

Table 2-8 Universe II MEM/IO Map - Jumper (E9)

Select	Jumper Position
Active	**In**
Disabled	Out

Table 2-9 CPU Clock Speed Selection - Jumper (E13)

Select	Jumper Position 333MHz	Jumper Position 450MHz
66 MHz	**In**	
100 MHz		**Out**

Table 2-10 Optional Fan Header- Jumper (E15)

Select	Pins
5V Fan	Pin 1 and 2

Table 2-10 Optional Fan Header- Jumper (E15)

Select	Pins
12V Fan	Pin 2 and 3

Table 2-11 CMOS Battery Enable - Jumper (E16)

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

Table 2-12 Watchdog Reset - Jumper (E18)

Select	Jumper Position
Active	In
Disabled	**Out**

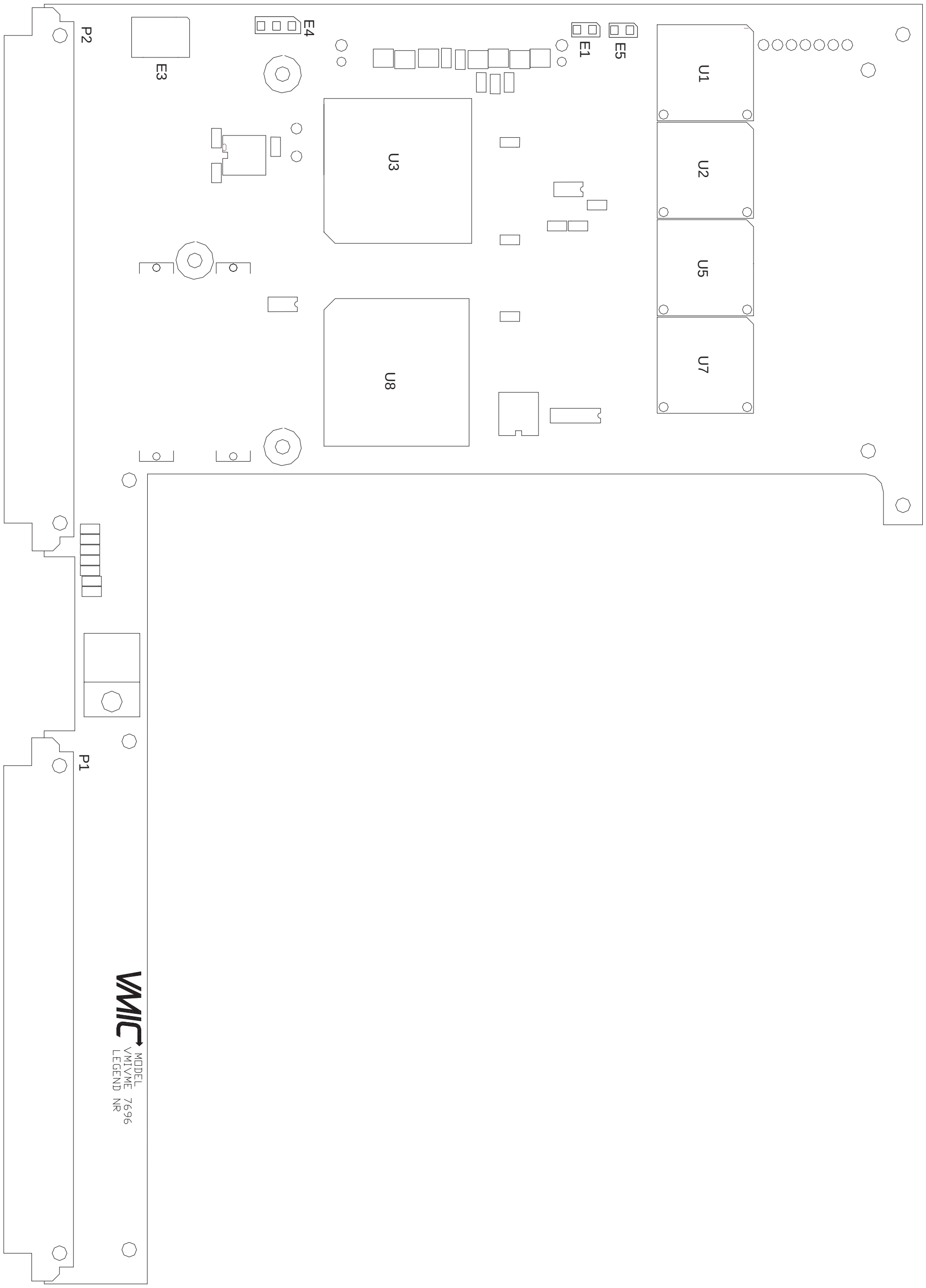


Figure 2-3 VMIVME-7696I Jumper Locations

The following connectors and jumpers are found on the VMIVME-7696I board. Default settings are in bold type.

Table 2-13 VMIVME-7696I Board Connectors

Connector	Function
J1 - J2	PMC Connectors
P1 - P2	VME Connectors
P3	Board-to-Board Connector
P4	Compact Flash Connector

Table 2-14 SCSI Speed Selection - Jumper (E1)

Select	Jumper Position
Ultra SCSI	In
Fast SCSI	**Out**

Table 2-15 EPLD In-circuit Programming Header (E3)

Select	Jumper Position
	Do Not Use

Table 2-16 NVRAM Battery Connection - Jumper (E4)

Select	Jumper Position
NVRAM Battery Active	**Pins 1 and 2**
NVRAM Battery Inactive	Pins 2 and 3

Installation

The VMIVME-7696 conforms to the VMEbus physical specification for a two slot 6U dual Eurocard (dual height). It can be plugged directly into any standard chassis accepting this type of board.

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

The following steps describe the VMIC recommended method for VMIVME-7696 installation and power-up:

1. Make sure power to the equipment is off.
2. Choose chassis slot. The VMIVME-7696 ***must*** be attached to a dual P1/P2 VMEbus backplane.

   If the VMIVME-7696 is to be the VMEbus system controller, choose the first two VMEbus slots. If some other board is the VMEbus system controller, choose any slot **except** slot one. The VMIVME-7696 does not require jumpers for enabling/disabling the system controller function.

Note

The VMIVME-7696 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-7696 board. For 20-slot VME configurations, three 100 CFM fans are recommended.

3. Insert the VMIVME-7696 and its attached expansion modules into the chosen VMEbus chassis slot (expansion modules should fill the slots immediately adjacent to the VMIVME-7696). 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-7696.
6. The VMIVME-7696 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-7696 peripheral driver software during operating system installation.

BIOS Setup

The VMIVME-7696 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-7696 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-7696 BIOS setup program are included in Appendix C.

Front Panel Connectors

The front panel connections, including connector pinouts and orientation, for the VMIVME-7696 are defined in detail in Appendix A.

PMC Expansion Site Connectors

The VMIVME-7696 supplies PMC expansion site connectors for adding a PMC expansion board. This expansion capability allows third-party devices to be used with the VMIVME-7696, as shown in Figure 2-4.

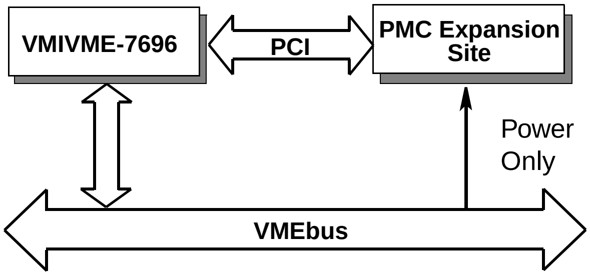
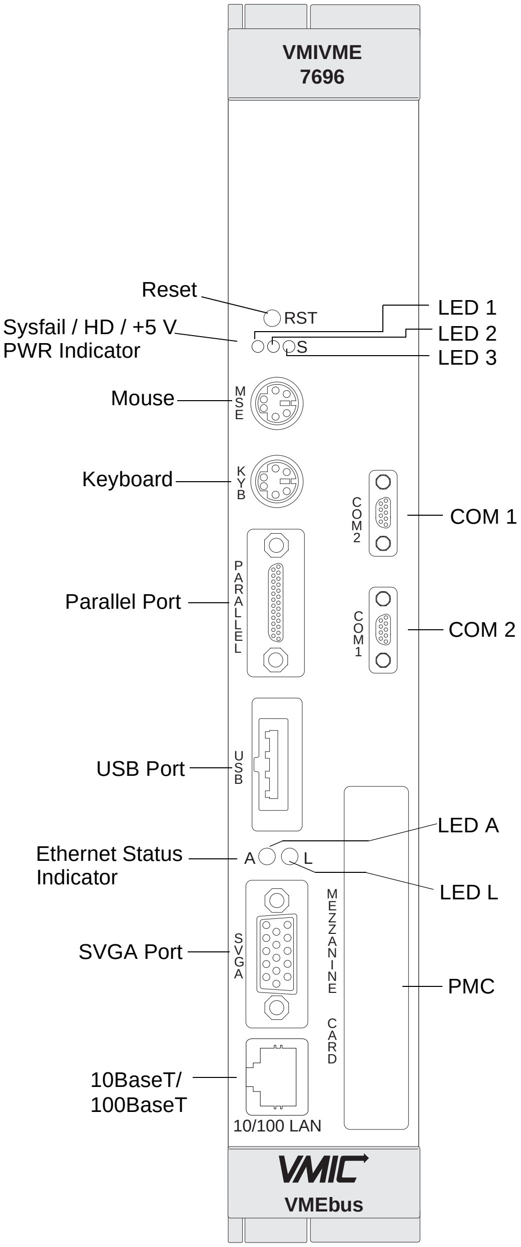


Figure 2-4 PCI Expansion Site

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 2-5 LED/Connector Positions on the Front Panel

CHAPTER

3

PC/AT Functions

Contents

Introduction

The VMIVME-7696 provides a complete IBM PC/AT-compatible Pentium II 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-7696 includes a PCI-compatible video adapter and Ethernet controller.

The following sections describe in detail the PC/AT functions of the VMIVME-7696.

CPU Socket

The VMIVME-7696 CPU socket is factory populated with a high-speed Pentium II processor. The CPU speed, SDRAM size and Compact Flash size are user-specified as part of the VMIVME-7696 ordering information. The options are not user-upgradable.

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

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

Physical Memory

The VMIVME-7696 provides Synchronous DRAM (SDRAM) as on-board system memory. Memory can be accessed as bytes, words, or longwords.

All RAM on the VMIVME-7696 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-7696 includes the system BIOS, video BIOS, SCSI BIOS, and LANWorks BIOS in a single flash memory device.

The VMIVME-7696 memory includes 128 Kbyte of battery-backed SRAM. 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.

Memory and Port Maps

Memory Map - Tundra Universe II-Based PCI-to-VMEbus Bridge

The memory map for the Tundra Universe II-based interface VMIVME-7696 is shown in Table 3-1. All systems share this same memory map, although a VMIVME-7696 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-7696, 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	Reserved
	$D8014 - $D8015	2 bytes	VMEBERR Address Modifier Register
	$D8016 - $D8017	2 bytes	Board IO Register (0x7696)
	$D8010 - $D8013	2 bytes	MEBERR Address Register
	$D800E - $D800F	2 bytes	System COMM Register
	$D8000 - $D800D	14 bytes	Reserved
	$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). BIOS will also map on-board PCI based NVRAM, Timers and Watchdog timers in this area.

** This space can be allocated as shared memory (for example, between the Pentium 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 Pentium II 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-7696 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-7696 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	PIIX4	Secondary Hard Disk Controller
$178 - $1EF	120		User I/O
$1F0 - $1F7	8	PIIX4	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 - $CFF			Reserved

* While these I/O ports are reserved for the listed functions, they are not implemented on the VMIVME-7696. 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 51 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-7696 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 56.

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-7696 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	
8	Real-Time Clock	
9	Old IRQ2	Determined by BIOS

Table 3-3 PC/AT Hardware Interrupt Line Assignments

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 56 depicts the VMIVME-7696 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

<table>
<tr><th colspan="2">Status Control Register
(I/O Address $061, Read/Write, Read Only)</th></tr>
<tr><td>Bit 7</td><td>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.</td></tr>
<tr><td>Bit 2</td><td>PCI SERR# Enable (Read / Write) - 1 = Clear and Disable, 0 = Enable</td></tr>
<tr><th colspan="2">Enable and Real-Time Clock Address Register
(I/O Address $070, Write Only)</th></tr>
<tr><td>Bit 7</td><td>NMI Enable - 1 = Disable, 0 = Enable</td></tr>
</table>

I/O Ports

The VMIVME-7696 incorporates the SMC Super-I/O chip. The SMC chip provides the VMIVME-7696 with a standard floppy drive controller, two 16550 UART-compatible serial ports, and one standard DB25 parallel port. The Ultra-IDE hard drive interface is provided by the Intel 82371AB (PIIX4) PCI ISA IDE Xcelerator chip. All ports are present in their standard PC/AT locations, using default interrupts.

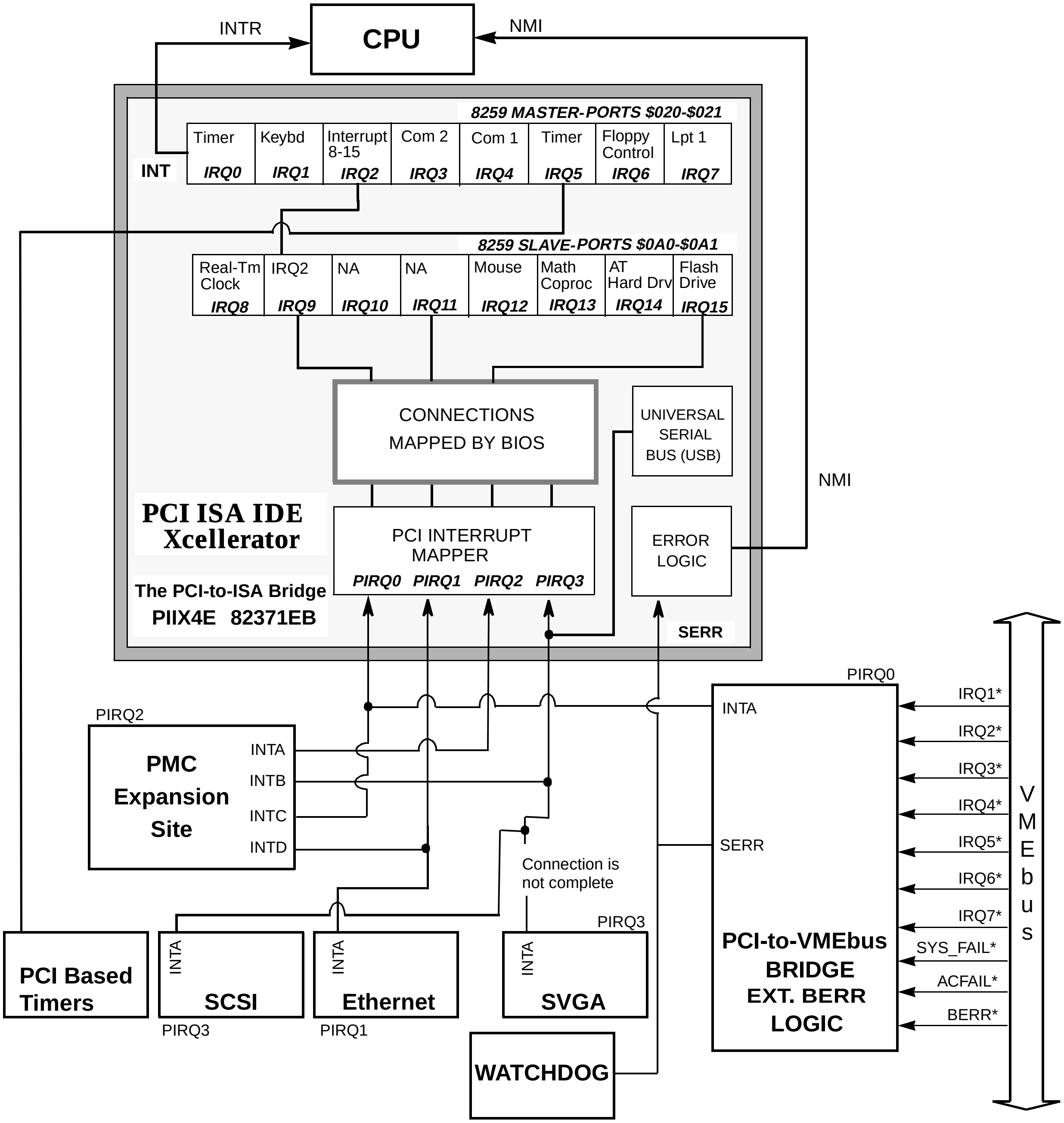


Figure 3-1 Connections for the PC Interrupt Logic Controller

Video Graphics Adapter

The monitor port on the VMIVME-7696 is controlled by a S3 Trio 3d AGP chip with 4 Mbyte video DRAM. The video controller chip is hardware and BIOS compatible with the IBM EGA and SVGA standards and also supports VESA high-resolution and extended video modes. Table 3-6 shows the graphics video modes supported by the Trio 3d video chip.

Table 3-6 Supported Graphics Video Resolutions

SCREEN RESOLUTION	MAXIMUM COLORS	MAXIMUM REFRESH RATES (Hz)
640 x 480	16 M	85
800 x 600	16 M	85
1,024 x 768	16 M	85
1,280 x 1,024	64 K	60
1600 x 1200	64 K	60

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.

The VMIVME-7696’s processor includes a 64-bit access to video memory with no-wait states. Video I/O registers are accessed using AGP bus.

The floppy disks supplied with the VMIVME-7696 contain drivers for Windows, Windows 95, and Windows NT (4.0) operating system. Appendix B contains instructions on the incorporation of the drivers during system installation.

Ethernet Controller

The network capability is provided by the Intel 21143. This Ethernet controller is PCI bus based and is software configurable. The VMIVME-7696 supports 10BaseT and 100BaseTx Ethernet. This ethernet supports LANWorks remote ethernet boot option. See Appendix C and D 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 theoretical maximum length of 100 m from the wiring hub to the terminal node.

100BaseTx

The VMIVME-7696 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 theoretical maximum deployment length of 250 m.

Universal Serial Bus

The VMIVME-7696 provides a single Universal Serial Bus (USB) connection on the front panel. The on-board USB controller completely supports the standard USB interface.

The USB Host Controller moves data between system memory and the USB by processing and scheduling data structures. The controller executes the scheduled lists, and reports status back to the system.

Note

Default CMOS settings of the VMIVME-7696 have USB functions disabled. This allows more interrupts and less Interrupt Latency for Real Time system. If USB is enabled, the user must be aware that Interrupt Sharing and Latency will be effected.

SCSI Controller

The Small Computer System Interface (SCSI) is provided on the VMIVME-7696. Ultra Fast Wide-SCSI II is implemented on the VMIVME-7696 system using Adaptec's AIC-7880 PCI-based SCSI controller. The SCSI architecture supports 8-bit (narrow) or 16-bit (wide) SCSI, for external SCSI devices. Ultra-SCSI provides faster disk access than a traditional disk interface, supporting a synchronous transfer rate of up to 40 Mbyte per second. In this manual, the Ultra / Fast / Wide SCSI is generically referred to as SCSI.

Internally, the SCSI bus is actively terminated; therefore, there are no peripherals located on the internal SCSI bus. The SCSI bus does provide external expansion to accommodate single-ended Ultra or Fast Wide SCSI devices such as external peripherals, plotters, scanners, and CD-ROM drives. Up to 15 external SCSI devices can be accommodated. The external Fast SCSI bus must be terminated at the last external SCSI device with an active, single-ended terminator.

The cable from the external devices attaches to the VMIVME-7696 at the VMEbus P2 connector. The SCSI connector and pinout is shown in Appendix A.

The VMIACC-0561 accessory plugs into the back of the P2 connector of the VMEbus backplane and provides transition of the SCSI signal from the P2 connector to a standard 68 pin SCSI connector.

CHAPTER 4

Embedded PC/RTOS Features

Contents

Introduction

VMIC’s VMIVME-7696 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-7696 also provides a bootable flash disk system, and 128 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.

The battery backed SRAM, general purpose timers and watchdog timers are part of a standard PCI Interface. Table 4-1 shows the PC Configuration Registers for these features.

Timers

There are many occasions in an industrial environment where the generation of accurate timing is required. The use of software alone to generate timing loops is awkward and wastes processor cycles. The hardware timers on-board the VMIVME-7696 are designed to offload from software the task of generating timing loops. Instead of generating software loops, the software engineer can configure each of the VMIVME-7696 timers to generate a periodic interrupt.

Table 4-1 PCI Configuration Space Registers

31	16	15	00	Register Address
Device ID 7696		Vendor ID 114A		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Latency Timer	Cache Line Size	0Ch
PCI Base Address 0 for Memory-Mapped Configuration Registers				10h
PCI Base Address 1 for I/O-Mapped Configuration Registers				14h
Reserved				18h
PCI Base Address for the 128 Kbyte NVRAM				1Ch
PCI Base Address for the Timers				20h
PCI Base Address for Watchdog Timer				24h
Reserved				28h
Reserved		Reserved		2Ch
Reserved				30h
Reserved				34h
Reserved				38h
Max_Lat	Min_gnt	Interrupt Pin	**Interrupt Line**	3Ch

Timer Structure

The VMIVME-7696 Timer segment contains three timers and a Timer Control section as illustrated in Figure 4-1 on page 64. Each timer is configured via software registers located within the Timer Control section. Each timer is set up to be either 32 or 16 bits wide depending on the required time duration. Furthermore, as illustrated in the block diagram, each timer consists of three different 16-bit counters (Scale Counter, Upper Counter, and Lower Counter).

The Timer Control section is the core of the Timer Structure. It contains the Timer Control circuitry, the Interrupt Status Block, and the Timer Control Registers. The Timer Control circuitry manages each timer and signals the Interrupt Control Block when a timer has timed out. The Interrupt Status Block maintains the Timer Interrupt Status (TIS) register and actually generates the interrupt to the host system (reference Table 4-7 on page 71). The Timer Control Registers contain all the registers which control the timer process.

There are three Timer Control Registers within the Timer Control Section:

- The *Timer Width/System State Registe*r determines whether the timers are 32 or 16 bits wide and also provides some board-level status based on the state of the jumpers.
- The *Timer Enable/Interrupt Register* enables counting within each timer and controls the Timer Interrupt masks.
- The *Timer Interrupt Status Register* provides the timer status.

Timer Functionality

Timer setup involves the following sequence:

- Setup timer width
- Load Counter Value into the timer
- Enable the timer

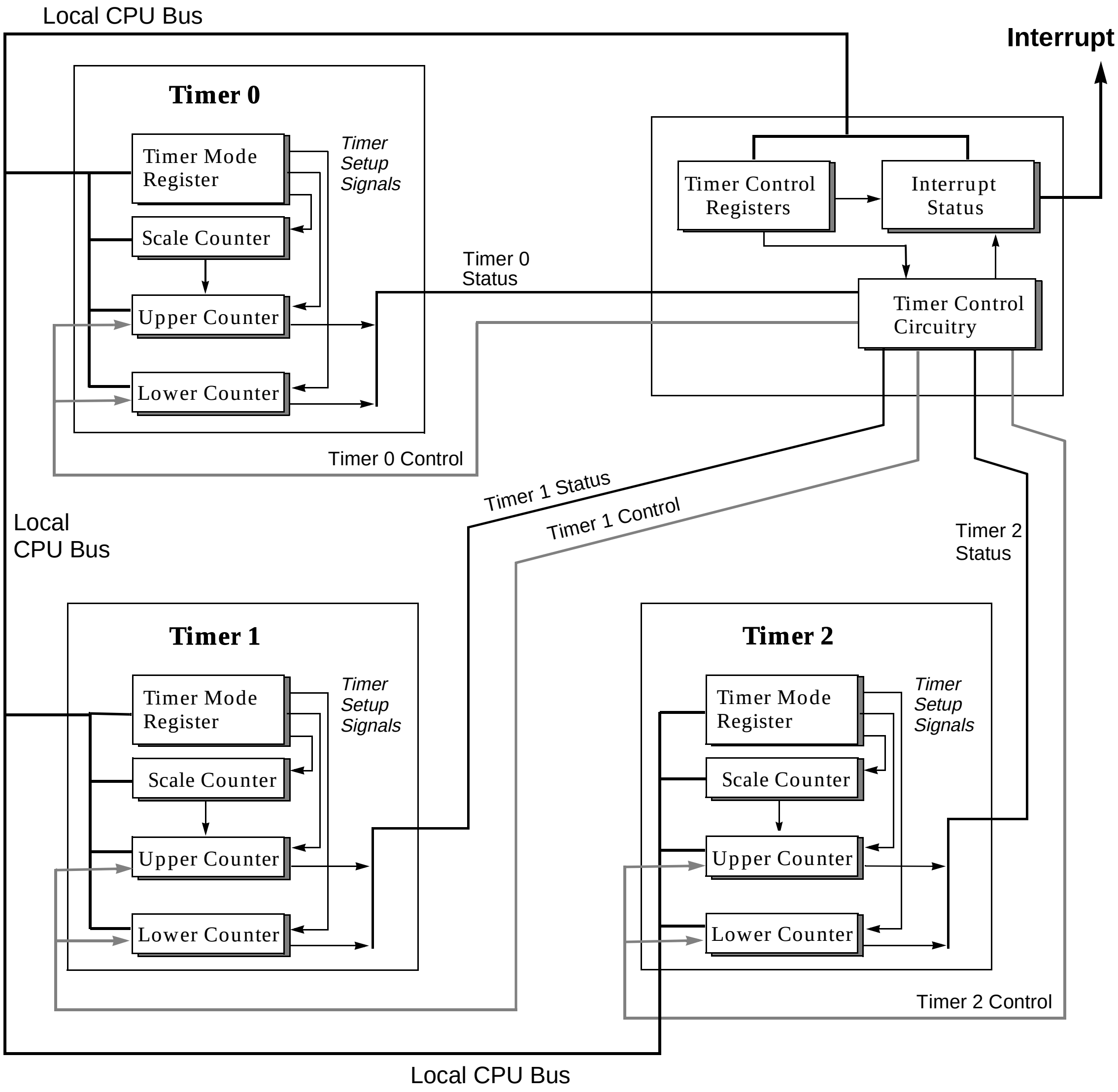


Figure 4-1 VMIVME-7696 Timer Block Diagram

The timers are set up by default to be 32 bits wide, but must be set to 16 bits wide for a time period of 65,538 µs or less. The Counter Value is loaded by first writing a Timer Mode byte (see Table 4-9 on page 73) to the timer and then writing the actual Counter Value to the timer. The Counter Value is defined in Table 4-10 on page 74 for 16-bit operation, or Table 4-11 on page 74 for 32-bit operation. After the mode and values are initialized, the timer is enabled.

At the end of the programmed time interval, the timer signals the Timer Control circuitry. The Timer Control circuitry sets a bit in the Timer Interrupt Status (TIS) register (reference Table 4-12 on page 75). The Timer Control circuitry then reinitializes the time to the original counter value and the timer starts counting again. The timer continues to loop in this fashion until disabled.

The host must read the TIS Register to determine the timer status. The host can choose to poll the TIS register or wait for an interrupt. An interrupt is available for use by the three timers. Each timer’s interrupt can be individually masked. The set bits and the pending interrupt are automatically cleared when the TIS register is read by the host. If more than one timer is enabled, more than one bit could be set in the TIS Register, but only one interrupt is issued.

Each timer is designed to provide an interrupt at repetitive time intervals based on the value placed in the timer. Each timer has a resolution of 1.0 µs. A value of x placed in the timer generates an interrupt every (x+2)* 1.0 µs. For example, a value of 8 placed in the timer has an cycle time of 10.0 µs ([8 + 2] * 1.0 µs).

Each timer has a cycle time range of between 3 µs to 71.58 minutes. See Table 4-2 for a more concise description of the Counter Values and their corresponding cycle times.

Table 4-2 Counter Value/Cycle Time Range Table (X is Counter Value)

Counter Value (HEX)		Cycle Time (µs)			
From	**To**	**From**	**To**	**Timer Width Setup**	**Description**
0001	0000	3	65,538	16 bits	Time Value is (X+2) µs.
0001 0001	0000 0000	65,539	4,295,032,834	32 bits	Time Value is (X+2) µs.

Table 4-3 Counter Value/Cycle Time Comparison Table

Counter Value (HEX)	Cycle Time (µs)	Timer Width Setup
0001	3	16-bit
FFFF	65,537	16-bit
0000	65,538	16-bit
0001 0001	65,539	32-bit
0001 FFFF	131,073	32-bit
0001 0000	131,074	32-bit

Table 4-3 Counter Value/Cycle Time Comparison Table (Continued)

Counter Value (HEX)	Cycle Time (µs)	Timer Width Setup
0002 0001	131,075	32-bit
0002 FFFF	196,609	32-bit
0002 0000	196,610	32-bit
FFFF 0001	4,294,901,763	32-bit
FFFF FFFF	4,294,967,297	32-bit
FFFF 0000	4,294,967,298	32-bit
0000 0001	4,294,967,299	32-bit
0000 FFFF	4,295,032,833	32-bit
0000 0000	4,295,032,834	32-bit

Note

The value 0001 (HEX) represents the shortest time duration; whereas, 0000 (HEX) represents the largest.

Polling

The VMIVME-7696 Timers can be used as polled timers. Two incidental characteristics of the timers must be kept in mind while polling. First, the timers, when counting in 16 or 32 bit mode, will always transition through an all 0xF state immediately prior to reinitialzation. For example, a typical count series for a 16 bit timer with an initial count of 0x0005 will be as follows:

State	Polled Count
6	0x0005
5	0x0004
4	0x0003
3	0x0002
2	0x0001
1	0x0000
0	0xFFFF
6	0x0005
5	0x0004

A typical series count for a 32 bit timer through the transition with an initial count of 0x00020140 will be as follows:

State	Polled Count
3	0x00000002
2	0x00000001
1	0x00000000
0	0xFFFFFFFF
131393	0x00020140
131392	0x0002013F

VMIC recommends that a value of one (1) be added to the polled value to obtain the correct count removing the all 0xF state. The 16 bit example with one (1) added to the polled value would be as follows:

State	Polled Count	Polled Count + 1
6	0x0005	0x0006
5	0x0004	0x0005
4	0x0003	0x0004
3	0x0002	0x0003
2	0x0001	0x0002
1	0x0000	0x0001
0	0xFFFF	0x0000
6	0x0005	0x0006
5	0x0004	0x0005

Timer Status

As previously mentioned, each timer is set to be polled or can be programmed to cause an interrupt as a result of timer expiration. Specific Upper and Lower Counters can be read to determine elapsed time since the previous read. However, the Timer/Interrupt Status register will most often be used to monitor timer activity.

The Timer Interrupt/Status register is used to clear timer interrupts as well as to determine timer rollover when interrupts are not being used. The Interrupt/Status bits are set when a timer has rolled over. If the specific timer is set up to cause interrupts, the action of the bit being set causes an interrupt. Timers continue to count after the rollover (expires).

Timer Interrupt/Status register bits are automatically cleared as a result of the read. A rollover can be detected (or interrupt cleared) without an additional write being required to *clear* the bit.

All bits that are read as a **1** during the register read are cleared. This is desirable since all three timers share a single interrupt on the PCI bus. If multiple timer channels are configured to cause interrupts, the Timer Interrupt Service routine must be written to handle multiple bits being set within the Timer Interrupt/Status register.

Timer Read-Back

Once enabled, the timers can be read to determine their present count. This is done by first issuing a Read-Back command to the particular timer’s TMR Register. The format of the Read-Back command is determined by whether the timer to be read is 32 or 16 bits wide (see Table 4-3). The Read-Back command latches the timer’s present count into an output register (the Lower Counter only if it is in 16-bit mode or both the Lower and Upper Counter if it is in 32-bit mode). The count is read by reading the Timer Counters, Upper and/or Lower, LSB first and then MSB.

Table 4-4 Read-Back Commands

Mode	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value in HEX
16-bit Mode	1	1	0	1	0	1	0	0	D4
32-bit Mode	1	1	0	1	1	1	0	0	DC

The latched count is retained in the output registers until they are read. If several Read-Back commands are issued to the same timer without reading the count, all but the first are ignored.

Table 4-4 shows an example sequence of reading the 16-bit count from Timer 0 set in 16-bit mode. When a timer is setup in 16 bit mode, only the Lower Counter is used.

Table 4-5 shows an example sequence of reading the 32-bit count from Timer 1 set in 32-bit mode. When a timer is set up in 32-bit mode, all three Counters, Upper, Lower, and Scale, are used. However, only the Lower and Upper Counters need to be read.

Table 4-5 16-bit Read/Mode Command Example

Step	Address offset (HEX)	Data (HEX)	Description
1	0C	D4	Write Read-Back command to Timer 0’s TMR Register
2	04	LSB	Read the LSB of Lower Counter
3	04	MSB	Read the MSB of Lower Counter

Table 4-6 32-bit Read/Mode Command Example

Step	Address offset (HEX)	Data (HEX)	Description
1	1C	DC	Write Read-Back command to Timer 1’s TMR Register
2	14	LSB	Read the LSB of the Lower Counter
3	14	MSB	Read the MSB of the Lower Counter
4	18	LSB	Read the LSB of the Upper Counter
5	18	MSB	Read the MSB of the Upper Counter

Timer Control Registers

The VMIVME-7696 Timer segment contains three timers and a Timer Control section, see Figure 4-1. Each timer is configured via software registers located within the Timer Control section. The Timer Control section is the core of the timer structure. It contains the Timer Control circuitry, the Interrupt Status Block, and the Timer Control Registers. The Timer Control Registers contain all the registers which control the timer process. These register are defined as:

- The *Timer Width/System State Registe*r *(Offset 30h)* determines whether the timers are 32 or 16 bits wide and also provides some board-level status based on the state of the jumpers.
- The *Timer Enable/Interrupt Register (Offset 34h)* enables counting within each timer and controls the Timer Interrupt masks.
- The *Timer Interrupt Status Register (Offset 38h)* provides the timer status.

A detailed description of the programming of these registers follows.

Programming

Upon powerup of the VMIVME-7696, the timers are in an undefined state. Each timer must be set up and enabled before it can be used. Each timer is completely independent of the others. Used timers do not need to be set up.

The VMIVME-7696 includes three timers. Each timer is implemented using an Intel 82C54 timer / counter chip. Each 82C54 contains three 16-bit counters. These counters are designated within the VMIVME-7696 as the Scale Counter, the Lower Counter, and the Upper Counter.

Table 4-7 defines the Timer Section Address Map. All offsets are based from the PCI memory base address for the timers. This base address is referred to a PCI base address and can be found in the VMIVME-7696 PCI Configuration Register space at offset 0x20 (reference Table 4-1). All data is transferred via the LSB (lower 8 bits) of the PCI data bus. All registers labeled *Register* are 8 bits wide. The Scale Counter, Lower Counter, and Upper Counter are 16 bits wide, and they are accessed using a byte-wide port using a defined sequence.

Depending on the desired cycle time, the upper counter and lower counter are cascaded to form a 32-bit timer. The cascading is controlled by the timer width register. In 32-bit mode, the scale counter is used to prescale the upper counter.

Table 4-7 Timer Section Address Map

Address Offset AD[5...0]	Segment	Description
00	Timer 0	Scale Counter (SC0)
04	Timer 0	Lower Counter (LC0)
08	Timer 0	Upper Counter (UC0)
0C	Timer 0	Timer Mode Register (TMR0)
10	Timer 1	Scale Counter (SC1)
14	Timer 1	Lower Counter (LC1)
18	Timer 1	Upper Counter (UC1)
1C	Timer 1	Timer Mode Register (TMR1)
20	Timer 2	Scale Counter (SC2)
24	Timer 2	Lower Counter (LC2)
28	Timer 2	Upper Counter (UC2)
2C	Timer 2	Timer Mode (TMR2) Register
30	Timer Control	Timer Width / System State (TWSS) Register
34	Timer Control	Timer Enable / Interrupt (TEI) Register
38	Timer Control	Timer Interrupt Status (TIS) Register
3C	Timer Control	Expansion ROM Read Enable Register

The timers have two width modes, 16 and 32-bit wide, the choice of which is determined by the Cycle Time required by the timer (see Table 4-3). For a Cycle Time of 65,538 µs or less, the timer is set to 16 bits wide. For a Cycle Time greater than 65,538 µs, the timer is set to 32 bits wide.

Note

The timer must be 16 bits wide to load a cycle time equal to or less than 65,538 µs (see Table 4-3).

The timer width is controlled by the Timer Width Bit Field (bits 2 to 0) of the Timer Width / System State (TWSS) Register (see Table 4-8). This register is located at offset 0x30 from the Timer PCI memory base address. Each of the bits correspond to one of the three timers. Bit 0 corresponds to Timer 0, Bit 1 corresponds to Timer 1, and Bit 2 corresponds to Timer 2. When the bit is set to a 1 (high), the timer is 32 bits wide, when the bit is set to a 0 (low), the timer is 16 bits wide. At powerup, the Timer Width Bits default to 32 bits wide state.

Timer Width Control/System State (TWSS) Register, Offset 30h

The Timer Width Control / System State (TWSS) Register, Offset 30h (Table 4-8) is used to provide board-level state information and control the width of the cascaded timers.

Table 4-8 Timer Width Control/System State (TWSS) Register, Offset 30h

Bit	Description	Read/ Write	Default
7	**Reserved**	Read Only	0
6	**Reserved**	Read Only	0
5	**Reserved**	Read Only	0
4 to 3	**Reserved**	Read Only	0
2	**Timer 2 Width:** Used by the Timer State Machine logic. Establishes whether the timer is 32 or 16 bits wide. Each bit corresponds to a timer. If the bit is set to a 1 (high), then the timer will be 32 bits wide. If the bit is set to a 0 (low), then the timer will be 16 bits wide.	R / W	1
1	Timer 1 Width	R / W	1
0	Timer 0 Width	R / W	1

The 16-bit width mode is implemented using one 16-bit counter of an 82C54 chip, specifically the Lower Counter. The 32-bit width mode is implemented using two 16-bit counters of an 82C54 chip. The most significant portion of the 32-bit timer is referred to as the Upper Counter, while the least significant portion is referred to as the Lower Counter. The Upper Counter is clocked using the Scale Counter.

A single timer mode register is used to control the modes and loading of the scale, lower and upper counter within a 82C54 timer.

Before each individual Counter is loaded with its Counter Value, a unique counter-specific control byte must be written to the Timer Mode Register (TMRx). Table 4-9 shows the Timer Mode Bytes. More specifically, before a Counter Value is loaded into the Scale Counter, the Scale Timer Mode byte (36) must be written to the Timer Mode Register. Before a Counter Value is loaded into the Lower Counter the Lower Timer Mode byte (7A) must be written to the Timer Mode Register, etc.

Table 4-9 Timer Mode Register Values

Counter	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value in HEX
Scale	0	0	1	1	0	1	1	0	36
Lower	0	1	1	1	1	0	1	0	7A
Upper	1	0	1	1	1	0	1	0	BA

The loading of the actual Counter Value is different based on whether the timer is 32 or 16 bits wide. For a timer setup to be 16 bits wide, a 16-bit Counter Value is loaded into the Lower Counter (LCx). For a timer setup to be 32 bits wide, the Lower Counter (LCx) is loaded with the least significant portion of the 32-bit Counter Value, while the Upper Counter (UCx) is loaded with the most significant portion of the 32-bit Counter Value. Also, when a timer is 32 bits wide, the Scale Counter (SCx) must be loaded with zeros.

Although each of the three Counters within an 82C54 Timer are 16 bits wide, they are addressed via a single 8-bit address location. To load a Counter Value into one of the Counters, one must write two bytes, representing the Least Significant Byte (LSB) and the Most Significant Byte (MSB) of a 16-bit Counter Value, to the same 8-bit address location.

Note

The Least Significant Byte must be written first and then the Most Significant Byte.

Table 4-10 shows an example sequence of setting up a Timer Mode Byte to the Timer Mode Register (TMR0, address offset 0C (HEX)) and writing a counter value of 0x45AD to the Lower Counter (LC0, address offset 04 (HEX)) of Timer 0. Timer 0 has previously been set to 16 bits wide.

Table 4-10 16-bit Wide Timer Counter Value Load Example

Step	Address Offset (HEX)	Data (HEX)	Description
1	0C	7A	Timer Mode Register (TMR0) byte setting up the Lower Counter of Timer 0.
2	04	AD	LSB byte of the counter value written to LC0.
3	04	45	MSB byte of the counter value written to LC0.

Table 4-11 shows an example sequence of loading a counter value to Timer 0, which has been previously set up to be 32 bits wide. First, the Scale Counter is loaded with zero. Then a counter value of 01BC 45AD (HEX) is loaded into the timer.

Table 4-11 32-bit Wide Timer Counter Value Load Example

Step	Address Offset (HEX)	Data (HEX)	Description
1	0C	36	Timer Mode Register (TMR0) byte setting up the Scale Counter 0 (SC0).
2	00	00	LSB byte with a value of zero written to the SC0.
3	00	00	MSB byte with a value of zero written to the SC0.
4	0C	7A	Timer Mode Register (TMR0) byte setting up the Lower Counter 0 (LC0).
5	04	AD	LSB byte of the LSW of the counter value written to the LC0.
6	04	45	MSB byte of the LSW of the counter value written to the LC0.
7	0C	BA	Timer Mode Register (TMR0) byte setting up the Upper Counter 0 (UC0).
8	08	BC	LSB byte of the MSW of the counter value written to the UC0.
9	08	01	MSB byte of the MSW of the counter value written to the UC0.

Timer Enable/Interrupt (TEI) Register: Offset 34h

The Final step in programming a timer involves setting up the Timer Interrupt Masks and enabling the timer. The Timer Enable / Interrupt (TEI) Register (Table 4-12) is used to do both of these tasks. It controls the turning on and off of each timer.

Bits 5 to 3 control the masking of the interrupt from each timer. When an Interrupt Mask bit is set to zero (0), the interrupt is not masked. When the bit is set to one (1), then the Timer Interrupt is masked.

Bits 2 to 0 of the Timer Enable / Interrupt (TEI) Register are the enable bits for each timer, respectively. Bit 0 enables Timer 0, Bit 1 enables Timer 1, etc. When the bit is set to zero (0), the timer is disabled. When the bit is set to a one (1), the timer is enabled.

Table 4-12 Timer Enable/Interrupt (TEI) Register: Offset 34h

Field	Description	Read/Write	Default
7 to 6	Reserved	R / W	0
5	Timer 2 Interrupt Mask	R / W	0
4	Timer 1 Interrupt Mask	R / W	0
3	Timer 0 Interrupt Mask	R / W	0
2	Enables Timer 2	R / W	0
1	Enables Timer 1	R / W	0
0	Enables Timer 0	R / W	0

Timer Interrupt Status (TIS) Register, Offset 38h

If more than one timer is enabled, the host needs to read the Timer Interrupt Status (TIS) Register (see Table 4-13) to determine which of the enabled timers counted through its count cycle. The TIS Register (Offset 38h address) displays which timer rolled through its count (expired). See Table 4-7 for additional address information. The register is cleared immediately upon being read. Bits 2 to 0 are the status bits for each timer, respectively. Bit 0 is the status bit for Timer 0, bit 1 is the status bit for Timer 1; the sequence continues.

Table 4-13 Timer Interrupt Status (TIS) Register, Offset 38h

Field	Description	Read/Write	Default
7 to 3	Reserved	Read Only	0
2	Timer 2 interrupt status	Read Only	0
1	Timer 1 interrupt status	Read Only	0
0	Timer 0 interrupt status	Read Only	0

Watchdog Timer

The VMIVME-7696 utilizes a Dallas DS1284 Watchdog Timekeeping Controller as its Watchdog Timer. The device provides a Time of Day feature and a Watchdog Alarm. The Time of Day feature found within the DS1284 device is explained in this section, but is not utilized by the VMIVME-7696. The actual Time of Day registers used by the VMIVME-7696 are located at the standard PC / AT I / O address. The Time of Day feature in the DS1284 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 Watchdog alarm can, under software control, generate a VME SYSFAIL or a Non-maskable Interrupt to the CPU. Bit 1 of the Watchdog RESET / SYSFAIL / SERR Routing Register and Bit 8 of the System COMM register (see Table 3-1) are used to enable this option. The System COMM Register is 2 bytes wide located at memory address $D800E. Bit 0 of the Watchdog RESET / SYSFAIL / SERR Routing Register is used to enable the NMI option (Table 4-14 on page 77).

Note

The Watchdog Timer Interrupt output must be set to Level Mode (see Watchdog Command Register bit 4) to use the SYSFAIL or SERR (NMI) option.

In addition if Bit 1 of the Watchdog RESET / SYSFAIL / SERR Routing Register is set the Watchdog Alarm is connected via a header (E18) to the CPU reset. The user can direct the Watchdog Alarm to reset the CPU if the jumper is installed.

Figure 4-2 shows a generalized block diagram of how the Watchdog Timer is used in the VMIVME-7696. The Watchdog Timer registers are memory-mapped in PCI space. Table 4-14 shows the Base Address Register of the Watchdog Timer.

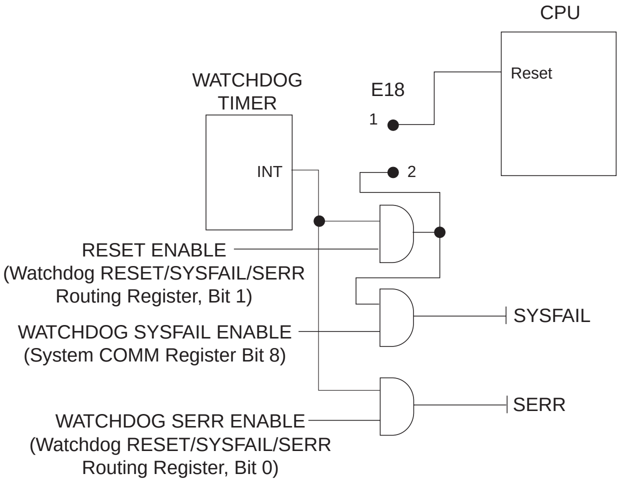


Figure 4-2 Watchdog Alarm Block

Table 4-14 Watchdog Registers

Register	Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Range
0	Base + 0	0.1 Seconds (BCD)				0.01 Seconds (BCD)				00 - 99
1	Base + 1	10 Seconds (BCD)				Seconds (BCD)				00 - 59
2	Base + 2	10 Minutes (BCD)				Minutes (BCD)				00 - 59
3	Base + 3	M	10 Minute Alarm (BCD)			Minute Alarm (BCD)				00 - 59
4	Base + 4	0	12/24	AM/PM*	10 Hr	Hours (BCD)				
5	Base + 5	M	12/24	AM/PM*	10 Hr	Hour Alarm (BCD)				
6	Base + 6	0	0	0	0	Days (BCD)				01 - 07
7	Base + 7	M	0	0	0	Day Alarm (BCD)				01 - 07
8	Base + 8	0	0	10 Date (BCD)		Date (BCD)				01 - 31
9	Base + 9	$\overline{\text{Eosc}}$	1**	0	10 Mo	Months (BCD)				01 - 12
A	Base + A	10 Years (BCD)				Years (BCD)				00 - 99
B	Base + B	Te	Ipsw	Ibh/lo	Pu/lvl	Wam	Tdm	Waf	Tdf	
C	Base + C	0.1 Seconds (BCD)				0.01 Seconds (BCD)				00 - 99
D	Base + D	10 Seconds (BCD)				Seconds (BCD)				00 - 99
E -3f	Base + (E -3f)	Scratch Pad Register (non-volatile)								
40	Base + 40	Watchdog RESET/SYSFAIL/SERR Routing Register						Bit 1=1[†]	Bit 0=1[‡]	

* 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.

[†] Bit 1=1 Watchdog output routs to RESET and VMEbus SYSFAIL logic.

[‡] Bit 0=1 Watchdog output routs to SERR, which is used to generate a Non-Maskable Interrupt.

Please note that only one bit (Bit 1 or Bit 0) may be set to a logic 1 at a time. Both bits set to logic 1 is an invalid setting.

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 while Bits 7 - 4 contain the 10 Minutes value with a range of 0 to 9 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. When AM Bit 5 is a zero (0) and when PM bit 5 is a 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 users 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 4-15). 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 4-15 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 J18 is loaded, the CPU will reset to a known state. Refer to Figure 4-2. 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 are 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 zeros 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 (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.

Watchdog Output Routing Register

The output of the Watchdog Timer can be routed to either reset the CPU, or cause a Non-Maskable Interrupt (NMI) via the Watchdog Output Routing Register (Base + 40).

If Bit 0 of this register is set to 1, the Watchdog Timer output will be routed to the Reset Jumper (J18). If Bit 1 of this register is set to 1, the Watchdog Timer output will be routed to the PCI SERR Line, to cause a Non-Maskable Interrupt.

Battery Backed SRAM

The VMIVME-7696 includes 128 K byte of battery-backed SRAM addressed in PCI memory space. Table 4-1 shows the PCI Base Address register (NVRAM, 1Ch) for the battery backed SRAM. 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.

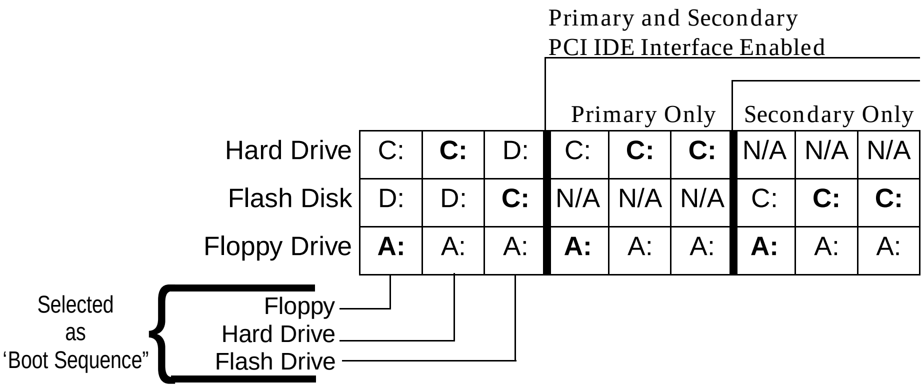
Flash Disk

The VMIVME-7696 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-7696 as the secondary IDE bus master device (the secondary IDE bus slave device is not assignable). The default setting in the AWARD 'STANDARD CMOS SETUP' screen is the 'AUTO' setting. In the AWARD 'INTERGRATED PERIPHERAL' screen, the on-chip secondary PCI IDE interface must be enabled for the flash disk to be functional. Refer to Appendix C, AWARD-Basic Input/Output System for additional details.

Figure 4-3 maps the configuration possibilities for a typical system consisting of the VMIVME-7696 with a resident flash disk, a hard drive attached to the Primary IDE interface, and a floppy drive attached to the floppy interface.



				Primary and Secondary PCI IDE Interface Enabled					
				Primary Only			Secondary Only		
Hard Drive	C:	**C:**	D:	C:	**C:**	**C:**	N/A	N/A	N/A
Flash Disk	D:	D:	**C:**	N/A	N/A	N/A	C:	**C:**	**C:**
Floppy Drive	**A:**	A:	A:	**A:**	A:	A:	**A:**	A:	A:
Selected as 'Boot Sequence'	Floppy	Hard Drive	Flash Drive						

Figure 4-3 Typical System Configuration

The Primary and Secondary PCI IDE Interfaces are controlled (enabled or disabled) in the Integrated Peripheral Setup screen of the AWARD BIOS. The First Boot Device is selected by the "Boot Sequence" line in the BIOS Features Setup screen.

Figure 4-3 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. Bootable being a device on which an operating system has been installed, or formatted as a system disk using MS-DOS.

Note

If during the configuration efforts a drive device is identified as a USER setting in the Standard CMOS Setup screen, as occurs after using the Auto-Detect Hard Drives function of the BIOS, then disabled (the device's IDE interface disabled) in the Integrated Peripheral Setup screen, the following error may occur:

Primary (or Secondary) Master HHD Error
Run Setup
Press <ESC> to Resume

Press the Escape key as directed to resume the boot procedure. The settings should be changed to prevent the reoccurrence of this error condition

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-7696 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. Data in the non-active partitions are not accessible during boot-up.

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-7696 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-7696, and enter CMOS Set-up.
2. Set Primary Master to “Not Installed”. Set Secondary Master to “Auto”.
3. Set boot device to floppy.
4. Boot DOS from the floppy, verify that the System Configuration Screen shows only the Flash Disk.
5. Run FDISK.
6. Delete all current partitions (any data currently stored in the partitions will be lost).
7. Exit FDISK, this will cause a reboot, then run FDISK again.
8. Create a primary partition.
9. 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.

10. Set the Primary partition as an active partition.
11. 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

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

17. Set boot device to desired boot source.

Drive letter assignments for a simple system were illustrated in Figure 4-3. 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.

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, contact VMIC for a Return Material Authorization (RMA) Number. **This RMA Number 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-7696 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-7696. Wherever possible, the VMIVME-7696 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.

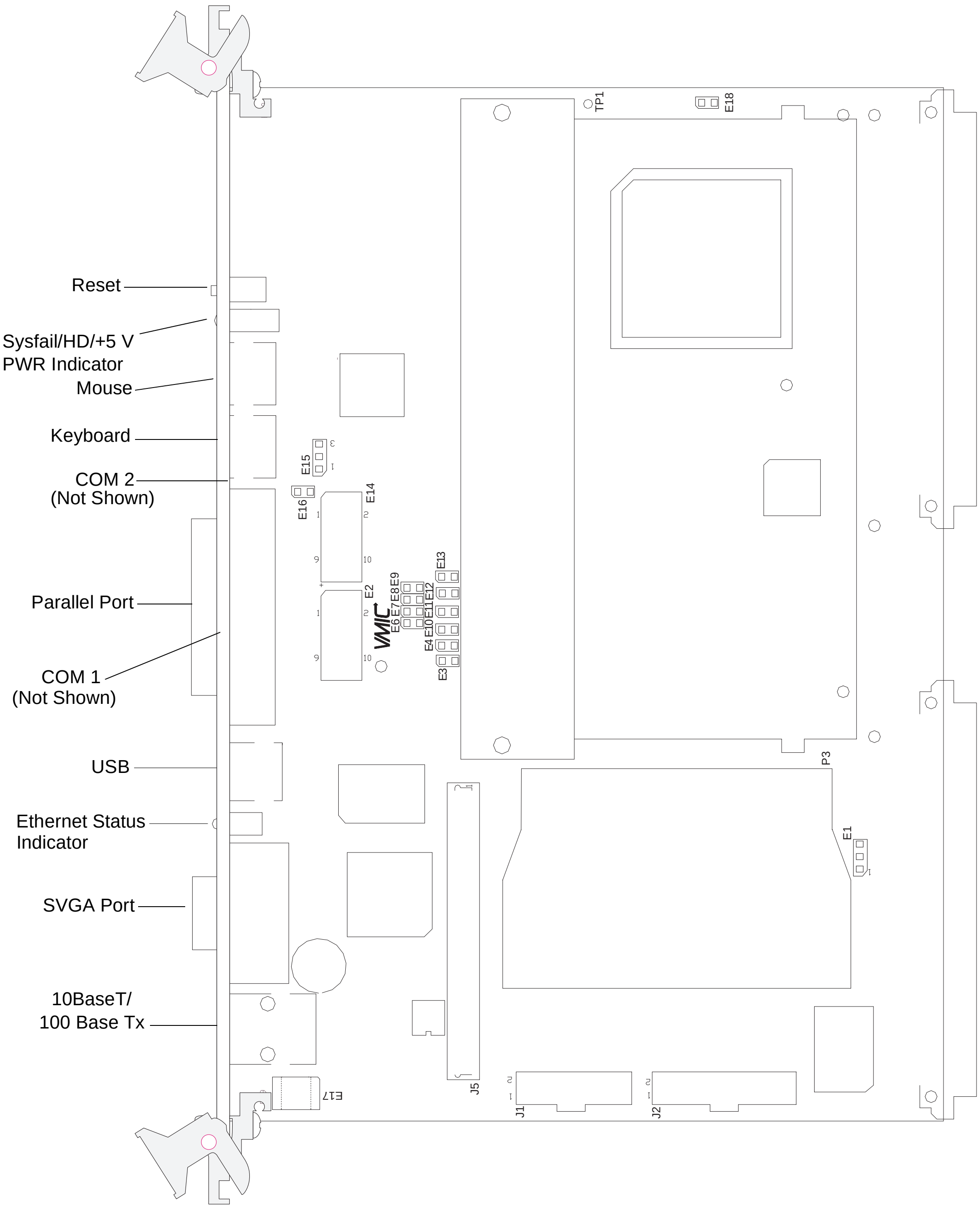
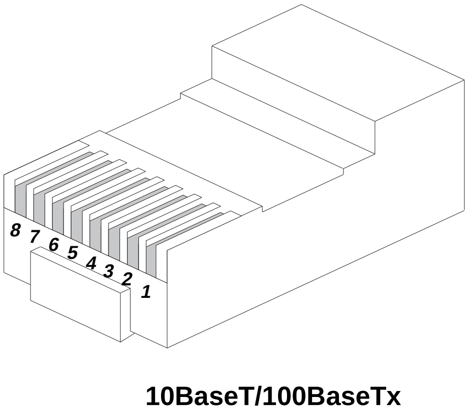


Figure A-1 VMIVME-7696 Connector Locations

Ethernet Connector Pinout

The pinout diagram for the Ethernet 10BaseT / 100BaseTx connector is shown in Figure A-2.

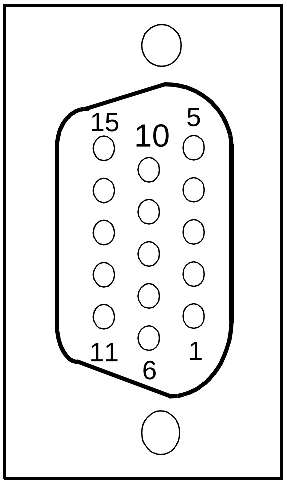


ETHERNET CONNECTOR 10BaseT/100BaseTx		
PIN	Signal Name	
1	TD+	Transmit Data
2	TD-	Transmit Data
3	RD+	Receive Data
4	NC	No Connection
5	NC	No Connection
6	RD-	Receive Data
7	NC	No Connection
8	NC	No Connection

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.

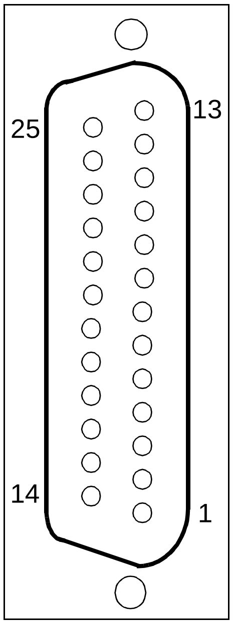


VIDEO CONNECTOR		
PIN	**DIRECTION**	**FUNCTION**
1	Out	Red
2	Out	Green
3	Out	Blue
4		Reserved
5		Ground
6		Ground
7		Ground
8		Ground
9		Reserved
10		Ground
11		Reserved
12		Reserved
13	Out	Horizontal Sync
14	Out	Vertical Sync
15		Reserved
Shield		Chassis Ground

Figure A-3 Video Connector Pinout

Parallel Port Connector Pinout

The parallel port shown in Figure A-4 uses a standard DB25 female connector typical of any PC / AT system.

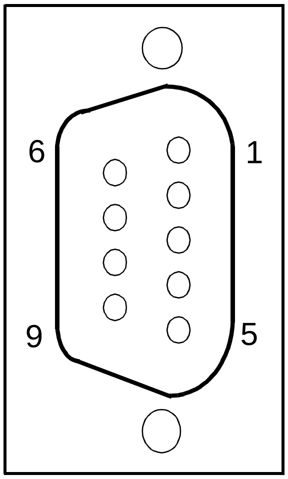


PARALLEL PORT CONNECTOR		
PIN	**DIRECTION**	**FUNCTION**
1	In / Out	Data Strobe
2	In / Out	Bidirectional Data D0
3	In / Out	Bidirectional Data D1
4	In / Out	Bidirectional Data D2
5	In / Out	Bidirectional Data D3
6	In / Out	Bidirectional Data D4
7	In / Out	Bidirectional Data D5
8	In / Out	Bidirectional Data D6
9	In / Out	Bidirectional Data D7
10	In	Acknowledge
11	In	Device Busy
12	In	Out of Paper
13	In	Device Selected
14	Out	Auto Feed
15	In	Error
16	Out	Initialize Device
17	In	Device Ready for Input
18		Signal Ground
19		Signal Ground
20		Signal Ground
21		Signal Ground
22		Signal Ground
23		Signal Ground
24		Signal Ground
25		Signal Ground
Shield		Chassis Ground

Figure A-4 Parallel Port Connector Pinout

Serial Connector Pinout

Each standard RS-232 serial port connector is a Microminiature D9 male as shown in the upper drawing in Figure A-5. Adapters to connect standard D9 serial peripherals to the board are available. Please refer to the product specification sheet for ordering information.

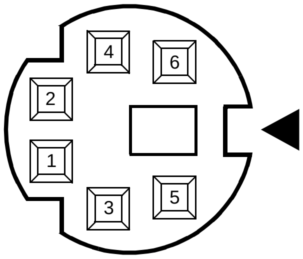


COM 1 and COM 2 SERIAL PORT CONNECTORS			
D9 PIN	DIR	RS-232 SIGNAL	FUNCTION
1*	In	DCD	Data Carrier Detect
2	In	RX	Receive Data
3	Out	TX	Transmit Data
4	Out	DTR	Data Terminal Ready
5		GND	Signal Ground
6	In	DSR	Data Set Ready
7	Out	RTS	Request to Send
8	In	CTS	Clear to Send
9*	In	RI	Ring Indicator
Shield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

Figure A-5 Serial Connector Pinouts

Keyboard Connector Pinout

The keyboard connector is a standard 6-pin female mini-DIN PS/2 connector as shown in Figure A-6.

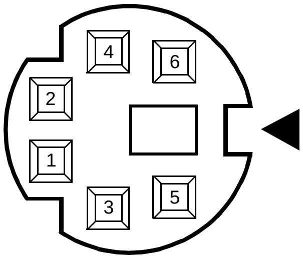


KEYBOARD CONNECTOR		
PIN	**DIR**	**FUNCTION**
1	In/Out	Data
2		Reserved
3		Ground
4		+5 V
5	Out	Clock
6		Reserved
Shield		Chassis Ground

Figure A-6 Keyboard Connector Pinout

Mouse Connector Pinout

The mouse connector is a standard 6-pin female mini-DIN PS/2 connector as shown in Figure A-7.



MOUSE CONNECTOR		
PIN	DIR	FUNCTION
1	In/Out	Data
2		Reserved
3		Ground
4		+5 V
5	Out	Clock
6		Reserved
Shield		Chassis Ground

Figure A-7 Mouse Connector Pinout

VMEbus Connector Pinout

Figure A-8 shows the location of the VMEbus P1 and P2 connectors and their orientation on the VMIVME-7696C (bottom board). Table A-1 shows the pin assignments for the VMEbus connectors. Note that only Row B of connector P2 is used; all other pins on P2 are reserved and should not be connected.

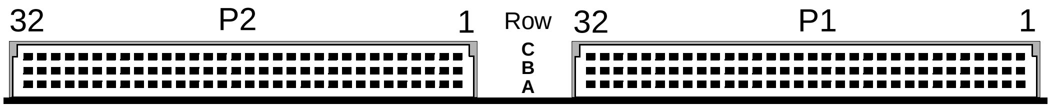


Figure A-8 VMEbus Connector Diagram

Table A-1 VMEbus Connector Pinout (bottom board)

PIN NUMBER	P1 ROW A SIGNAL	P1 ROW B SIGNAL	P1 ROW C SIGNAL	P2 ROW A SIGNAL	P2 ROW B SIGNAL	P2 ROW C SIGNAL
1	D00	BBSY	D08	GND	+5 V	IDE RST#
2	D01	BCLR	D09	DDP8	GND	DDP7
3	D02	ACFAIL	D10	DDP9	Reserved	DDP6
4	D03	BG0IN	D11	DDP10	A24	DDP5
5	D04	BG0OUT	D12	DDP11	A25	DDP4
6	D05	BG1IN	D13	DDP12	A26	DDP3
7	D06	BG1OUT	D14	DDP13	A27	DDP2
8	D07	BG2IN	D15	DDP14	A28	DDP1
9	GND	BG2OUT	GND	DDP15	A29	DDP0
10	SYSCLK	BG3IN	SYSFAIL	IDE REQ0	A30	IOCS1.64#
11	GND	BG3OUT	BERR	IDE IOW0 #	A31	GND
12	DS1	BR0	SYSRESET	IDE IOR0 #	GND	GND
13	DS0	BR1	LWORD	IDE IORDY0#	+5 V	GND
14	WRITE	BR2	AM5	HD_ACT #	D16	IDESELA
15	GND	BR3	A23	GND	D17	IDE DACK0#
16	DTACK	AM0	A22	GND	D18	IDE IRQ0
17	GND	AM1	A21	DAP1	D19	DAP 2

Table A-1 VMEbus Connector Pinout (bottom board) (Continued)

PIN NUMBER	P1 ROW A SIGNAL	P1 ROW B SIGNAL	P1 ROW C SIGNAL	P2 ROW A SIGNAL	P2 ROW B SIGNAL	P2 ROW C SIGNAL
18	AS	AM2	A20	IDECS01 #	D20	DAP 0
19	GND	AM3	A19	GND	D21	IDE CS03#
20	IACK	GND	A18	DRATED	D22	REDWC
21	IACKIN	SERCLK	A17	GND	D23	INDEX#
22	IACKOUT	SERDAT	A16	DRVSB #	GND	MOTEA#
23	AM4	GND	A15	GND	D24	DRUSA#
24	A07	IRQ7	A14	GND	D25	MOTEB#
25	A06	IRQ6	A13	GND	D26	STEP#
26	A05	IRQ5	A12	GND	D27	WD4TA#
27	A04	IRQ4	A11	GND	D28	TRK#
28	A03	IRQ3	A10	GND	D29	RDATA#
29	A02	IRQ2	A09	DSKCHG #	D30	SIDE1#
30	A01	IRQ1	A08	GND	D31	DIR
31	-12 V	+5 V STDBY	+12 V	VCC	GND	WGATE
32	+5 V	+5 V	+5 V	VCC	+5 V	WPT

Figure A-9 shows the location of the VMEbus P1 and P2 connectors and their orientation on the VMIVME-7696I (top board). Table A-2 shows the pin assignments for the VMEbus connectors. Note that only Row B of connector P2 is used; all other pins on P2 are reserved and should not be connected.

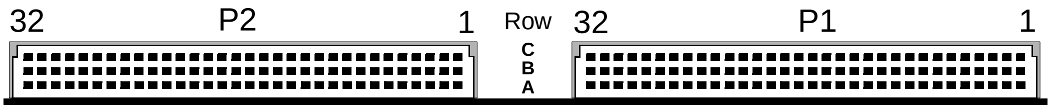


Figure A-9 VMEbus Connector Diagram

Table A-2 VMEbus Connector Pinout (top board)

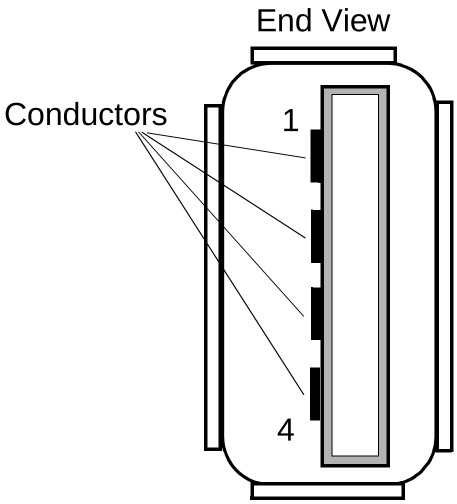
PIN NUMBER	P1 ROW A SIGNAL	P1 ROW B SIGNAL	P1 ROW C SIGNAL	P2 ROW A SIGNAL	P2 ROW B SIGNAL	P2 ROW C SIGNAL
1	NA	NA	NA	SSCD 12	+5 V	GND
2	NA	NA	NA	GND	GND	SSCD 13
3	NA	NA	NA	SSCD 14	NA	GND
4	NA	BG0IN	NA	GND	NA	SSCD 15
5	NA	BG0OUT	NA	SSCD PH#	NA	GND
6	NA	BG1IN	NA	SSCD 0	NA	SSCD 1
7	NA	BG1OUT	NA	GND	NA	GND
8	NA	BG2IN	NA	SSCD 2	NA	SSCD 3
9	GND	BG2OUT	GND	GND	NA	GND
10	NA	BG3IN	NA	SSCD 4	NA	SSCD 5
11	GND	BG3OUT	NA	GND	NA	GND
12	NA	NA	NA	SSCD 6	GND	SSCD 7
13	NA	NA	NA	GND	+5 V	GND
14	NA	NA	NA	SSCD PL #	NA	SATN #
15	GND	NA	NA	GND	NA	GND
16	NA	NA	NA	SBSY #	NA	SACK #
17	GND	NA	NA	GND	NA	GND
18	NA	NA	NA	SRESET #	NA	SMSG #

Table A-2 VMEbus Connector Pinout (top board) (Continued)

PIN NUMBER	P1 ROW A SIGNAL	P1 ROW B SIGNAL	P1 ROW C SIGNAL	P2 ROW A SIGNAL	P2 ROW B SIGNAL	P2 ROW C SIGNAL
19	GND	NA	NA	GND	NA	GND
20	NA	GND	NA	SSEL #	NA	SCD #
21	IACKIN	NA	NA	GND	NA	GND
22	IACKOUT	NA	NA	SREQ #	GND	SIO
23	NA	GND	NA	GND	NA	GND
24	NA	NA	NA	NA	NA	TERM PWR
25	NA	NA	NA	NA	NA	NA
26	NA	NA	NA	GND	NA	SSCD 9
27	NA	NA	NA	SSCD 8	NA	GND
28	NA	NA	NA	GND	NA	SSCD 11
29	NA	NA	NA	SSCD 10	NA	NA
30	NA	NA	NA	NA	NA	Reserved
31	-12 V	NA	+12 V	+5	GND	NA
32	+5 V	NA	+5 V	+5	+5 V	Reserved

USB Connector

The USB port uses an industry standard 4 position shielded connector. Figure A-10 shows the pinout of the USB connector.



USB CONNECTOR		
PIN	SIGNAL	FUNCTION
1	USBV	USB Power
2	USB-	USB Data -
3	USB+	USB Data +
4	USBG	USB Ground

Figure A-10 USB Connector Pinout

Appendix

B

System Driver Software

Contents

Introduction

The VMIVME-7696 provides high-performance video, and Local Area Network (LAN) access by means of on-board PCI and AGP-based adapters and associated software drivers. The APG-based video adapter used on the VMIVME-7696 is the S3 Trio 3d. High-performance LAN operation including 10BaseT and 100BaseTx, is provided by the Intel 21143 Ethernet controller chip.

To optimize performance of each of these PCI-based subsystems, the VMIVME-7696 is provided with software drivers compatible with DOS, Windows for Workgroups Version 3.11, 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-7696, 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 for Workgroups Version 3.11, Windows 95, or Windows NT 4.0 operating systems are described in the following sections.

Windows for Workgroups (Version 3.11)

1. Format the IDE hard drive and install MS-DOS.
2. Begin installation of Windows for Workgroups 3.11, following the instructions provided by Microsoft for Express Setup. When you reach the 'NETWORK SETUP' screen, the statement 'No Network Installed' will be displayed.

Note

If you do not require LAN operation, click 'CONTINUE' and skip steps 3 through 12.

3. The following steps will load the Digital Semiconductor Ethernet drivers which are used to configure the Intel 21143 adapter. From the main 'NETWORK SETUP' screen, click on the 'NETWORKS' button.
4. The 'Networks' screen should be displayed. Choose 'Install Microsoft Windows Network' option.
5. Then click 'OK'.
6. The 'NETWORK SETUP' screen appears again with the option for 'SHARING'. Click on 'SHARING' and choose the options that fit your system requirements by placing an 'X' in the boxes shown. Then click 'OK'.
7. Again at the 'NETWORK SETUP' screen, click 'CONTINUE'.
8. Under 'ADD NETWORK ADAPTER' click on 'UNLISTED or UPDATED NETWORK ADAPTER'. Then click 'OK'.
9. Insert the VMIVME-7696 distribution disk marked 320-500022-005 into drive A: and type: **`A:\32bit\WFW311\`**. Then click 'OK'.
10. Under 'UNLISTED' or 'UPDATED NETWORK ADAPTER', choose 'Digital Semiconductor 21143-based 10/100 mpbs Ethernet Controller (ND153)'. Then click 'OK'.
11. Under 'MICROSOFT WINDOWS NETWORK NAMES', enter the network names you want for computer name, group name, etc. Click 'OK'.
12. Windows for Workgroups should now continue with regular installation. During the remaining installation steps, use the full path name, A:\32bit\WFW311\, whenever prompted for the Digital Semiconductor 21143 driver diskette.
13. After Windows for Workgroups installation is complete, you may choose to install the S3 video drivers. If you do not require the S3 video drivers for operation, please skip steps 14 through 24.
14. Insert the VMIVME-7696 distribution diskette marked 320-500022-001 into drive A:
15. From the Program Manager Screen double-click the mouse on the 'Windows Setup' icon.

16. Under the Windows setup screen select Options. Click on ‘Change System Setup’.
17. Click on the display line. Select ‘Other Display (requires Disk from OEM’.
18. In the Windows Setup window type **`A:\32bit\`** then select ‘OK’.
19. Select the correct driver from the list and click ‘OK’.
20. When the ‘Installation Complete’ screen is displayed click ‘OK’.
21. After the files are installed, the installation utilities boots WinMode which will allow the selection of the system monitor type.
22. In the ‘WINMODE’ screen, adjust the monitor settings as required and then click ‘OK’.
23. Click ‘OK’ to restart windows.
24. In the ‘VGA UTILITY’ window, there will be a ‘WINMODE’ icon that will allow the user to adjust the VGA display.
25. The network must be setup properly. It is necessary to setup the proper connection type for your system.

Proceed with the following steps to set your network connection type.

Note

If you are not running a network, please skip steps 26 through 30.

26. From the ‘PROGRAM MANAGER’ screen, double-click on the ‘NETWORK’ icon.
27. Then double-click on ‘NETWORK SETUP’ icon.
28. Under ‘NETWORK SETUP’, double-click on ‘Digital Semiconductor 21143-based 10/100 mpbs Ethernet Controller (ND153)’.
29. Under ‘ADVANCED NETWORK-ADAPTER SETTINGS’, choose ‘CONNECTION TYPE’ and choose the ‘CONNECTION TYPE’ value of ‘*AUTO SENSE TYPE*’. Click on ‘SET’ and then ‘OK’.
30. Under ‘NETWORK SETUP’, click on ‘OK’. If the network connection type has changed, click ‘OK’ at the next screen for the information message about ‘SYSTEM.INI’ and ‘PROTOCOL.INI’ and click on ‘RESTART COMPUTER’ for the new network settings to take effect.

The unit should now be configured for operation in the WINDOWS FOR WORKGROUPS 3.11 environment.

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. When you reach the 'WINDOWS 95 SETUP WIZARD SCREEN', choose 'TYPICAL' under 'SETUP OPTIONS' and then click on 'NEXT'.
4. When you reach the 'ANALYZING YOUR COMPUTER' screen, place an 'X' in the box for 'NETWORK ADAPTER', then click on 'NEXT'.
5. Under the 'WINDOWS COMPONENTS SCREEN', select 'INSTALL THE MOST COMMON COMPONENTS' and then click on 'NEXT'.
6. Continue with the installation until Windows 95 is completely installed and has rebooted.

Note

If you do not require LAN operation, skip steps 7 through 25.

7. From the main Windows 95 screen, click on 'START'.
8. Click on 'SETTINGS' and then 'CONTROL PANEL'.
9. Double-click on the 'SYSTEM' icon and select the 'DEVICE MANAGER' tab.
10. Double-click on 'OTHER DEVICES' and then double-click on 'PCI ETHERNET CONTROLLER'.
11. Select the 'DRIVER' tab and then select 'CHANGE DRIVER'.
12. In the 'SELECT HARDWARE TYPE' screen, choose 'NETWORK ADAPTERS' and click on 'OK'.
13. Insert the Diskette marked 320-500022-005 into drive A:
14. In the 'SELECT DEVICE' window, click on 'HAVE DISK' and type **`A:\32bit\WIN95\INF`** and then click 'OK'.
15. Under 'SELECT DEVICES' choose 'Digital Semiconductor 21143-BASED 10/100 MPBS ETHERNET CONTROLLER', then click 'OK'.
16. Under 'PCI ETHERNET CONTROL PROPERTIES' select 'OK'. The system will prompt you for computer and workgroup names. Type in the names you wish to use in the spaces shown and then choose 'CLOSE'.
17. An 'INSERT DISK' window will be displayed, click on 'OK'.
18. Under the 'COPYING FILES' window type **`A:\32bit\win95\`**. Click on 'OK'.
19. An 'INSERT DISK' window will be displayed, click on 'OK'.
20. Under the 'COPYING FILES' window type the directory where Windows 95 is located. For example, if you are loading from a CD-ROM located at D:\, the desired response is D:\WIN95.

21. At the ‘SYSTEM PROPERTIES’ window, click ‘OK’.
22. From ‘CONTROL PANEL’, double-click on the ‘NETWORK’ icon.
23. Under ‘NETWORK’, click on ‘FILE AND PRINT SHARING’ and choose the appropriate items for your system, click ‘OK’.
24. At the ‘NETWORK’ window, click ‘OK’. When prompted, insert the diskettes needed to complete the network installation, if required.
25. When the system prompts you to restart your computer, click on ‘YES’ for the network settings to take effect.
26. From the main Windows 95 screen, click on ‘START’.
27. Next, click on ‘SETTINGS’ and the ‘CONTROL PANEL’.
28. Double-click on the ‘DISPLAY’ icon and select the ‘SETTINGS’ tab.
29. A ‘DISPLAY’ window appears prompting for the use of Hardware Installation Wizard, click on ‘Cancel’.
30. Click on ‘CHANGE DISLAY TYPE’, then click on ‘CHANGE’ in the ‘ADAPTER TYPE’ field.
31. Select ‘S3’ as the manufacturer, then click ‘HAVE DISK’.
32. Insert the diskette labeled 320-500022-003 into drive A:. Type ‘**`A:\`**’ as the files source (if not already displayed) and click on ‘OK’.
33. Under ‘SELECT DEVICE’, choose ‘S3 Trio 3d’, then click ‘OK’.
34. When the CHOOSE DISPLAY TYPE’ screen returns, click on ‘CLOSE’.
35. When the ‘Display Properties” screen returns, click on ‘CLOSE’.
36. Restart the computer if prompted to allow the new settings to take effect.

SCSI Driver Installation Directions For Windows 95

The SCSI drivers may be installed as follows:

1. From Windows 95, click the mouse on ‘START’, then ‘SHUTDOWN’.
2. Select ‘RESTART THE COMPUTER IN MS-DOS MODE’, then click on ‘YES’.
3. Wait for the C:\WINDOWS prompt, then insert the Diskette marked 320-500022-08 into drive A:
4. Type `A:` <ENTER>. Wait for the A:\ prompt, then type **`INSTALL`** <ENTER>.
5. Follow the instructions for default installation of the SCSI drivers.
6. When the installation is complete, remove the diskette from the drive, and press CTRL-ALT-DEL to reboot the computer.

The unit should now be properly configured for operation in Windows 95.

Windows 95 INF Update Utility for Intel(TM) Chipsets

This update allows the operating system to correctly identify the Intel(TM) chipset components and properly configure the system.

1. Windows 95 must be fully installed and running on the system prior to running this software.
2. Close any running applications. You may experience some difficulties if you don't close all applications.
3. Insert the Diskette marked 320-500022-004 into drive A: and run A:\95\INF_UP\SETUP.EXE.
4. Click Next on Welcome Screen to read the license agreement.
5. Click Yes if you agree to continue. If you click No, the program will terminate.
6. Click Next in the Installer screen.
7. Click OK to restart.
8. Follow the screen instructions and use the default settings to complete the setup when Windows 95 is re-started.

Bus Master IDE Driver for Windows 95

1. Ensure the system is operating correctly.
2. Close any running applications.
3. The driver files are stored in an integrated application setup program. This program is a Windows 95 program that allows the driver files to be INSTALLED or DE-INSTALLED. Insert the Diskette marked 320-500022-004 into drive A: and run A:\95\BM_IDE\SETUP.EXE.
4. Select 'Next' on Welcome Screen to continue.
5. View the 'Software Use and Distribution License Agreement' and choose 'Yes' if you agree to continue. If you choose 'No' the program will terminate.
6. Next, select 'INSTALL' to install the Intel(r) BM-IDE Driver.

Note

If the driver is currently installed on the system, SETUP will ask you whether or not you want to continue. Follow the prompts on the screen to Install the driver if desired.

7. Select 'OK' to restart the system when prompted to do so.
8. Follow the screen instructions and use default settings to complete the setup when Windows 95 is re-started.

Upon re-start, Windows 95 will display that it has found Intel PCI Bus Master IDE controller hardware and is installing hardware for it.

9. Select ’Yes’ when prompted to re-start Windows 95.

After installation, the following driver and related files are stored as listed.

- <Windows 95 directory>\SYSTEM\IOSUBSYS\IDEATAPI.MPD
- <Windows 95 directory>\SYSTEM\IOSUBSYS\PIIXVSD.VXD
- <Windows 95 directory>\INF\IDEATAPI.INF

Note

This driver is not to be used with Windows 98. The setup program must be rerun to uninstall the driver if Windows 98 is to be used.

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 on the ‘SELECT FROM LIST’ button.
5. Click on the ‘HAVE DISK’ button.
6. Insert disk 320-500022-008 into drive A:.
7. Type **`A:\32bit\wnt40\ndis40`** and click ‘OK’.
8. In the ‘SELECT OEM OPTION’, choose ‘Digital Semiconductor 21143-based 10/100 mpbs Ethernet Controller (NDIS3)’, then click ‘OK’.
9. Select the above entry on the displayed list, click on ‘NEXT’.
10. Select the NetBEUI Protocol (only), click on ‘NEXT’.
11. Click on ‘NEXT’ to install selected components.
12. Click ‘CONTINUE’ to allow an Autosense connection type.
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 on ‘OK’ to continue.
15. In the ‘DISPLAY PROPERTIES’ window, click on ‘TEST’.

Note

Please note that Windows NT 4.0 does not allow the selection of the S3 drivers during initial setup.

If the display test is successful, click on ‘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 on ‘RESTART COMPUTER’.

Note

Service PACK #3 must be installed.

17. When the computer reboots, double-click on ‘MY COMPUTER’ window.
18. Double-click on the ‘CONTROL PANEL’ icon in the ‘MY COMPUTER’ window.
19. Double-click on the ‘DISPLAY’ icon in the ‘CONTROL PANEL’.
20. Select the ‘SETTINGS’ tab in the ‘DISPLAY PROPERTIES’ window, then click on the ‘DISPLAY TYPE’ button.
21. In the ‘DISPLAY TYPE’ window, click on ‘CHANGE’.
22. In the ‘CHANGE DISPLAY’ window, click on ‘HAVE DISK’.
23. Insert disk 320-500022-002 into drive A:
24. Type **`A:\32bit`** and click ‘OK’.
25. ‘S3 Trio 3d’ will be displayed in the ‘CHANGE DISPLAY’ window. Click on ‘OK’.
26. Proceed as directed, removing the driver disk from the floppy drive, and restart the computer to activate the new settings. When the system reboots, the ‘INVALID DISPLAY SETTINGS’ screen will be displayed. Click on ‘OK’.
27. On the ‘DISPLAY PROPERTIES’ screen click on ‘SETTINGS’, then click on ‘TEST’.
28. The ‘TESTING MODE’ screen will be displayed. Click on ‘OK’. If the bitmap test image is displayed correctly, click on ‘OK’.

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

Appendix

C

Award - BIOS

Contents

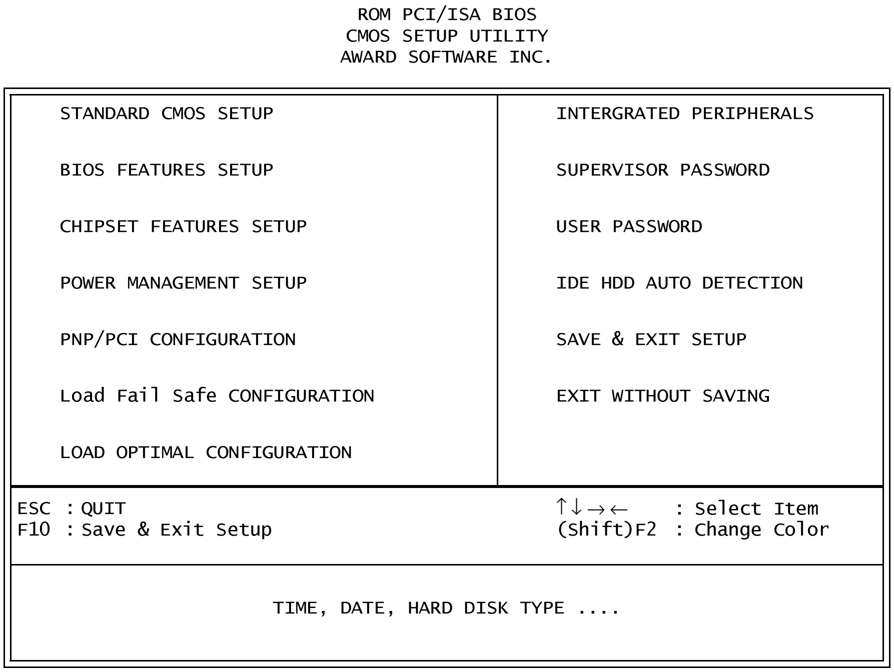
Introduction

The VMIVME-7696 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-7696 BIOS.

System BIOS Setup Utility

During system bootup, press the Delete key to access the Award *Elite*BIOS CMOS Setup Utility screen. From this screen, the user can select any section of the Award (system) BIOS for configuration, such as floppy drive configuration or system memory.

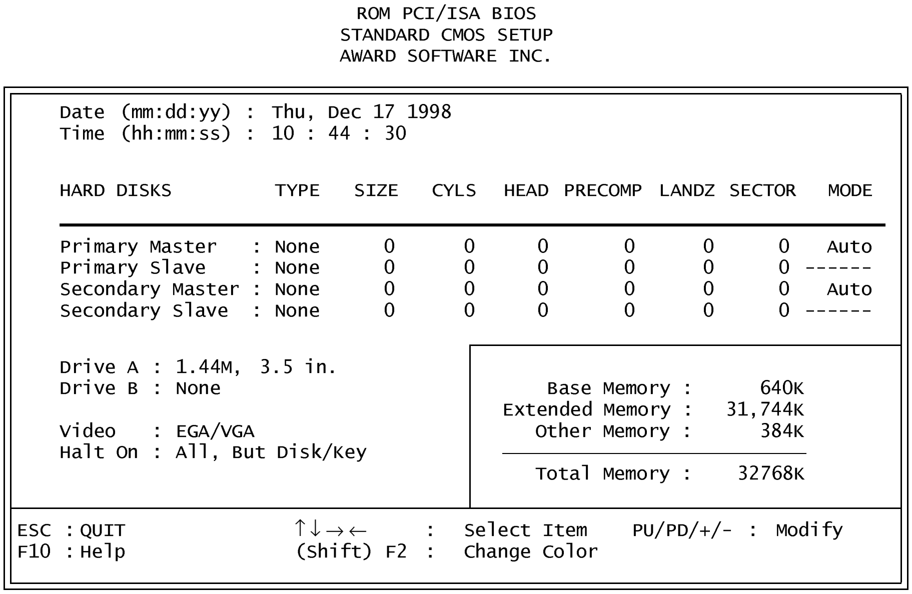
The parameters shown throughout this section are the default values.



```
                              ROM PCI/ISA BIOS
                             CMOS SETUP UTILITY
                            AWARD SOFTWARE INC.

    STANDARD CMOS SETUP                   INTERGRATED PERIPHERALS

    BIOS FEATURES SETUP                   SUPERVISOR PASSWORD

    CHIPSET FEATURES SETUP                USER PASSWORD

    POWER MANAGEMENT SETUP                IDE HDD AUTO DETECTION

    PNP/PCI CONFIGURATION                 SAVE & EXIT SETUP

    Load Fail Safe CONFIGURATION          EXIT WITHOUT SAVING

    LOAD OPTIMAL CONFIGURATION

ESC  : QUIT                               ↑↓→←       : Select Item
F10  : Save & Exit Setup                  (Shift)F2  : Change Color

                    TIME, DATE, HARD DISK TYPE ....
```

Standard CMOS Setup

Selection of the first main menu item, the Standard CMOS Setup, allows the user to set the system clock and calendar, record disk drive parameters, video subsystem type, and select the type of errors that will cause a system halt.



```
                    ROM PCI/ISA BIOS
                  STANDARD CMOS SETUP
                  AWARD SOFTWARE INC.

    Date  (mm:dd:yy) :  Thu, Dec 17 1998
    Time  (hh:mm:ss) :  10 : 44 : 30

    HARD DISKS         TYPE   SIZE   CYLS   HEAD  PRECOMP  LANDZ  SECTOR   MODE
    ------------------------------------------------------------------------------
    Primary Master   : None      0      0      0       0      0      0   Auto
    Primary Slave    : None      0      0      0       0      0      0 ------
    Secondary Master : None      0      0      0       0      0      0   Auto
    Secondary Slave  : None      0      0      0       0      0      0 ------

    Drive A : 1.44M,  3.5 in.
    Drive B : None                              Base Memory :      640K
                                            Extended Memory :   31,744K
    Video   : EGA/VGA                          Other Memory :      384K
    Halt On : All, But Disk/Key              ---------------------------
                                               Total Memory :    32768K

ESC : QUIT              ↑↓→←      :  Select Item    PU/PD/+/-  :  Modify
F10 : Help              (Shift) F2  :  Change Color
```

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. Note that the day of the week field is for information purposes only, and will be set based on the other date information.

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.

Primary Master/Slave

The VMIVME-7696 has the capability of utilizing one IDE hard disk drive on the Primary Master bus. The default setting is None. The Primary Slave is not used with the VMIVME-7696.

Secondary Master/Slave

The Secondary Master is the resident flash disk. The default setting is None. The Secondary Slave is not assignable.

Floppy Disk Drive

Floppy Drive A

The VMIVME-7696 supports one floppy disk drive. The options are:

• None	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-7696 does not support a second floppy drive. The default is None.

Video

The VMIVME-7696 has an EGA/VGA graphics chip onboard. The BIOS supports a secondary video subsystem, but it is not selected in Setup. The default is EGA/VGA. Use the PgUp or PgDn key to select the video.

Halt On

During the power-on self-test (POST), the computer may be made to halt if the BIOS detects a hardware error. The options are:

• All Errors	If the BIOS detects any non-fatal error, POST stops and prompts you for corrective action
• No Errors	POST does not stop for any errors
• All, But Keyboard	Post does not stop for a keyboard error, but stops for all other errors
• All, But Diskette	POST does not stop for diskette drive errors, but stops for all other errors
• All, But Disk/Key	POST does not stop for a keyboard or disk error, but stops for all other errors (default)

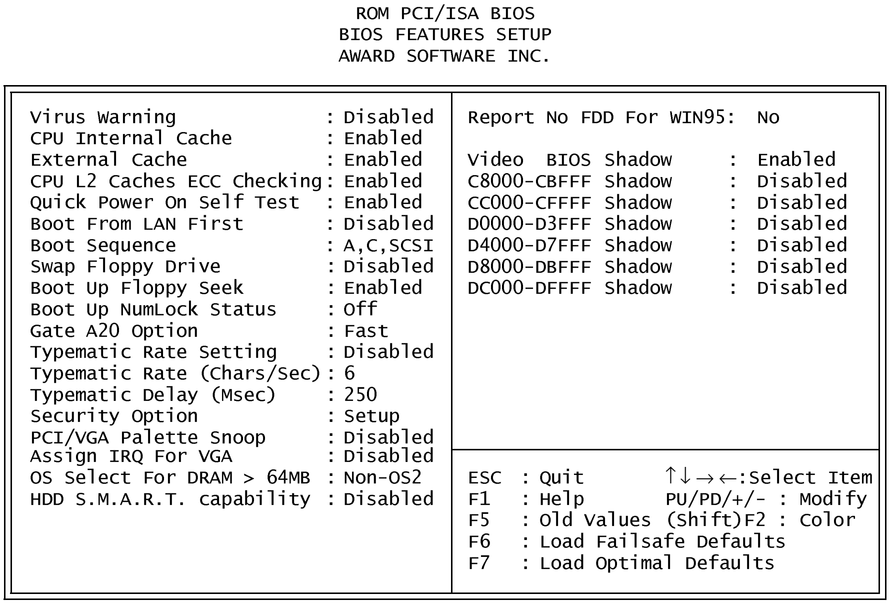
Use the PgUp or PgDn key to select the errors which will halt the system. The default is All, But Disk/Key.

Memory

The Memory field at the lower right of the screen is for informational purposes only and can not be modified by the user. This field displays the total RAM installed in the system, and the amounts allocated to base, extended, and other (high) memory.

BIOS Features Setup

This screen, selected from the CMOS Setup Utility screen, allows the user to configure options that are in addition to the basic BIOS features.



```
                          ROM PCI/ISA BIOS
                        BIOS FEATURES SETUP
                        AWARD SOFTWARE INC.

 Virus Warning             : Disabled   Report No FDD For WIN95:  No
 CPU Internal Cache        : Enabled
 External Cache            : Enabled    Video  BIOS Shadow     :  Enabled
 CPU L2 Caches ECC Checking: Enabled    C8000-CBFFF Shadow     :  Disabled
 Quick Power On Self Test  : Enabled    CC000-CFFFF Shadow     :  Disabled
 Boot From LAN First       : Disabled   D0000-D3FFF Shadow     :  Disabled
 Boot Sequence             : A,C,SCSI   D4000-D7FFF Shadow     :  Disabled
 Swap Floppy Drive         : Disabled   D8000-DBFFF Shadow     :  Disabled
 Boot Up Floppy Seek       : Enabled    DC000-DFFFF Shadow     :  Disabled
 Boot Up NumLock Status    : Off
 Gate A20 Option           : Fast
 Typematic Rate Setting    : Disabled
 Typematic Rate (Chars/Sec): 6
 Typematic Delay (Msec)    : 250
 Security Option           : Setup
 PCI/VGA Palette Snoop     : Disabled
 Assign IRQ For VGA        : Disabled
 OS Select For DRAM > 64MB : Non-OS2    ESC  : Quit      ↑↓→←:Select Item
 HDD S.M.A.R.T. capability : Disabled   F1   : Help      PU/PD/+/- : Modify
                                        F5   : Old Values (Shift)F2 : Color
                                        F6   : Load Failsafe Defaults
                                        F7   : Load Optimal Defaults
```

Virus Warning

When enabled, you receive a warning message if a program (specifically, a virus) attempts to write to the boot sector or the partition table of the hard disk drive. You should then run an anti-virus program. Keep in mind that this feature protects only the boot sector, not the entire hard drive. The default is Disabled.

Note

NOTE: Many disk diagnostic programs that access the boot sector table can trigger the virus warning message. If you plan to run such a program, we recommend that you first disable the virus warning.

CPU Internal Cache

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 were it is stored until processed by the CPU. The default is Enabled.

External Cache

The external cache (up to 512K) provides additional cache for use by the CPU. This cache functions in the same manner as internal cache. The default is Enabled.

CPU L2 Cache ECC Checking

When you select Enabled, memory checking is enable when the external cache contains ECC SRAMs. The default is Enabled.

Quick Power On Self Test

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

Boot From LAN First

When enabled, this option allows the CPU to boot off of a connected network. The default is Disabled.

Boot Sequence

Determines the order in which the BIOS will seek a bootable drive. The default order is the floppy disk (A:), then the internal hard drive (C:), followed by the SCSI drive.

Swap Floppy Drive

The option is functional only in a system with two floppy drives. The VMIVME-7696 supports only one floppy drive. Changing this option will have no effect on the system. The default is Disabled.

Boot Up Floppy Seek

When enabled, the BIOS will test the floppy to determine if the floppy drive has 40 or 80 tracks. Only 360K floppy drives have 40 tracks. It is recommended that this option be set to disabled unless a third party 360K floppy drive is being used in the system. The default is Enabled.

Boot Up NumLock Status

Toggle between On or Off to control the state of the NumLock key when the system boots. When toggled On, the numeric keypad generates numbers instead of controlling the cursor operations. The default is Off.

Gate A20 Option

Gate A20 refers to the way the system addresses memory above 1 MB (extended memory). When set to Fast, the system chipset controls Gate A20. When set to Normal, a pin in the keyboard controller controls Gate A20. Setting Gate A20 to Fast improves system speed, particularly with OS/2 and Windows. The default setting is Fast.

Typematic Rate Setting

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

Typematic Rate (Chars/Sec)

If the Typematic Rate Setting is enabled this determines the rate a character is repeated when a key is held down. The options are: 6, 8, 10, 12, 15, 20, 24, or 30 characters per second.

Typematic Delay (Msec)

If the Typematic rate Setting is enabled this determines the delay before a character starts repeating when a key is held down. The options are: 250, 500, 750, or 1000 milliseconds.

Security Option

If a password has been set, this determines whether the password is required every time the system boots, or only when the Setup is accessed. The default is Setup.

PCI/VGA Palette Snoop

Enabling the video palette snoop allows the PMC add-in card to share a common palette with the on-board graphics controller. The default is Disabled.

Assign IRQ for VGA

Select Enabled only if your VGA card requires an assigned IRQ. Most ordinary cards do not; some high-end cards with video capture function do. Consult the information that comes with your VGA card to determine whether it needs an assigned IRQ. The default is Disabled

OS Select For DRAM>64MB

Select OS2 only if you are running OS/2 operating system with greater than 64 MB of RAM on the system. The default is Non-OS2.

HDD S.M.A.R.T. Capability

SMART is an acronym for Self-Monitoring Analysis and Reporting Technology system. SMART is a hard drive self-diagnostic feature available on some IDE hard drives. The default is Disabled.

Report No FDD For WIN95

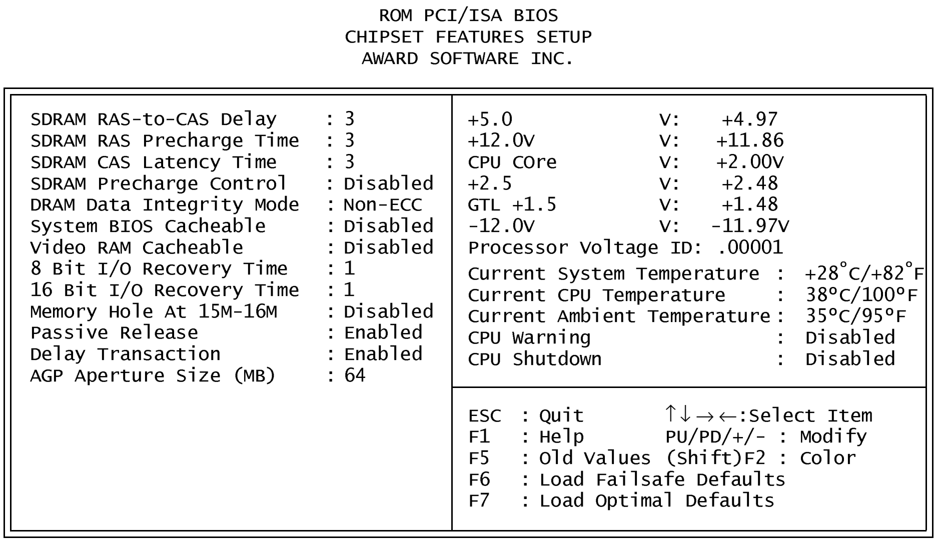
Select *Yes* to release IRQ6 when the system contains no floppy drive, for compatibility with Windows 95 logo certification. In the Integrated Peripherals screen, select *Disabled* for the Onboard FDC Controller field. The default is No.

Chipset Features Setup

This section describes features of the Intel 82430TX PCIset.

Advanced Options

The parameters in this screen are for system designers, service personnel, and technically competent users only. Do not reset these values without a complete understanding of the consequences.



```
                     ROM PCI/ISA BIOS
                  CHIPSET FEATURES SETUP
                   AWARD SOFTWARE INC.

SDRAM RAS-to-CAS Delay     : 3          +5.0            V:    +4.97
SDRAM RAS Precharge Time   : 3          +12.0V          V:   +11.86
SDRAM CAS Latency Time     : 3          CPU COre        V:   +2.00V
SDRAM Precharge Control    : Disabled   +2.5            V:    +2.48
DRAM Data Integrity Mode   : Non-ECC    GTL +1.5        V:    +1.48
System BIOS Cacheable      : Disabled   -12.0V          V:   -11.97V
Video RAM Cacheable        : Disabled   Processor Voltage ID: .00001
8 Bit I/O Recovery Time    : 1          Current System Temperature :  +28°C/+82°F
16 Bit I/O Recovery Time   : 1          Current CPU Temperature    :  38°C/100°F
Memory Hole At 15M-16M     : Disabled   Current Ambient Temperature:  35°C/95°F
Passive Release            : Enabled    CPU Warning                :  Disabled
Delay Transaction          : Enabled    CPU Shutdown               :  Disabled
AGP Aperture Size (MB)     : 64
                                        ESC  : Quit        ↑↓→←:Select Item
                                        F1   : Help        PU/PD/+/- : Modify
                                        F5   : Old Values  (Shift)F2 : Color
                                        F6   : Load Failsafe Defaults
                                        F7   : Load Optimal Defaults
```

SDRAM(CAS Lat/RAS-to-CAS)

You can select a combination of CAS latency and RAS-to-CAS delay in HCLKs of 2 or 3. The values in this field depend on the DRAM installed. Do not change the values in this field unless you change specifications of the installed DRAM or the installed CPU. The default is 3.

SDRAM RAS Precharge Time

The precharge time is the number of cycles it takes for the RAS to accumulate its charge before DRAM refresh. If insufficient time is allowed, refresh may be incomplete and the DRAM may fail to retain data. This field applies only if synchronous DRAM is installed in the system. The default is 3.

SDRAM CAS Latency Time

When synchronous DRAM is installed, you can control the number of CLKs between when the SDRAMs sample a read command and when the controller samples read data from the SDRAMs. Do not reset this field from the default value specified by the system designer. The default is 3.

SDRAM Precharge Control

When Enabled, all CPU cycles to SDRAM result in an All Banks Precharge Command on the SDRAM interface. The default is Disabled.

DRAM Data Integrity Mode

Select Non-ECC or ECC (error-correcting code), according to the type of installed DRAM. The default is Non-ECC

System BIOS Cacheable

Selecting Enabled allows caching of the system BIOS ROM at F0000h-FFFFFh, resulting in better system performance. However, if any program writes to this memory area, a system error may result. The default is Disabled

Video RAM Cacheable

Selecting Enabled allows caching of the video BIOS ROM at C0000h to C7FFFh. If a program writes to this memory area, a system error may occur. The default is Disabled.

8 Bit I/O Recovery Time

The I/O recovery mechanism adds bus clock cycles between PCI-originated I/O cycles to the ISA bus. This delay takes place because the PCI bus is faster than the ISA bus. The default is 1 (one).

16 Bit I/O Recovery Time

The I/O recovery mechanism adds bus clock cycles between PCI-originated I/O cycles to the ISA bus. This delay takes place because the PCI bus is faster than the ISA bus. The default is 1.

These two fields (8 Bit and 16 Bit) let you add recovery time (in bus clock cycles) for 8-bit and 16-bit I/O. The default is 1.

Memory Hole at 15M-16M

This area of system memory may be reserved for ISA adapter ROM. When this area is reserved, it cannot be cached. Refer to the documentation that came with the peripheral that require the use of this area of system memory for memory requirements. The default is Disabled.

Passive Release

When Enabled, CPU to PCI bus accesses are allowed during passive release. Otherwise, the arbiter only accepts another PCI master access to local DRAM. The default is Enabled.

Delayed Transaction

The chipset has an embedded 32-bit posted write buffer to support delay transactions cycles. Select Enabled to support compliance with PCI specification version 2.1. The default is Enabled.

AGP Aperture Size (MB)

Select the size of the Accelerated Graphics Port (AGP) aperture. The aperture is a portion of the PCI memory address range dedicated for graphics memory address space. Host cycles that hit the aperture range are forwarded to the AGP without any translation. See http://www.agpforum.org for APG information. The default is 64.

CPU Warning Temperature

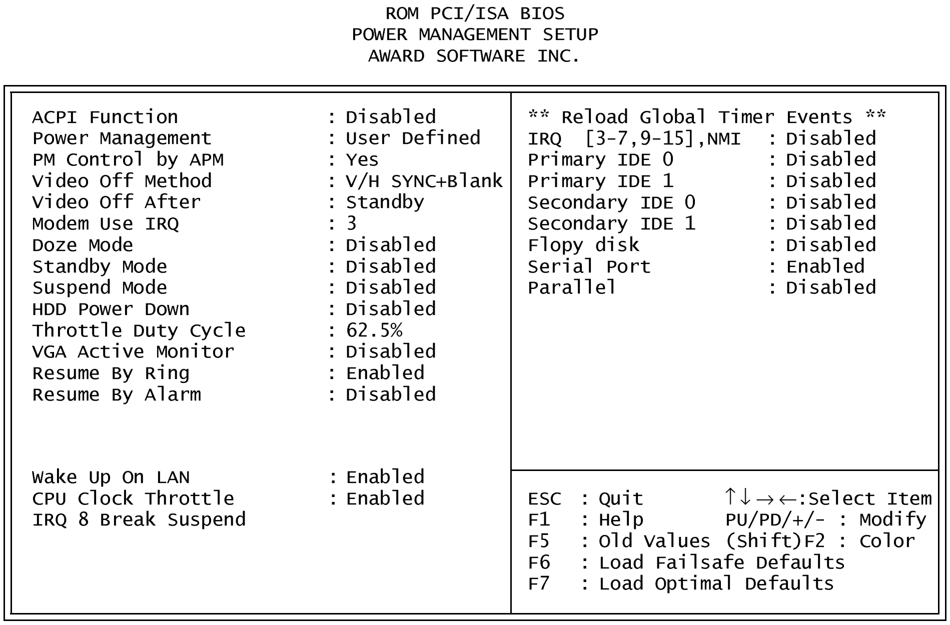
Select the combination of lower and upper limits for the CPU temperature. If the CPU temperature extends beyond either limit, any warning mechanism programmed into your system will be activated. The default is Disabled.

CPU Shutdown

When enabled this feature will throttle the CPU speed down when the temperature exceeds the preset limit. The percentage the CPU will be throttled is set in the Power Management screen by the Throttle Duty Cycle. The default is Disabled.

Power Management

This section discusses the power management features provided with the installed Award BIOS.



```
                        ROM PCI/ISA BIOS
                     POWER MANAGEMENT SETUP
                      AWARD SOFTWARE INC.

ACPI Function             : Disabled        ** Reload Global Timer Events **
Power Management          : User Defined    IRQ  [3-7,9-15],NMI  : Disabled
PM Control by APM         : Yes             Primary IDE 0        : Disabled
Video Off Method          : V/H SYNC+Blank  Primary IDE 1        : Disabled
Video Off After           : Standby         Secondary IDE 0      : Disabled
Modem Use IRQ             : 3               Secondary IDE 1      : Disabled
Doze Mode                 : Disabled        Flopy disk           : Disabled
Standby Mode              : Disabled        Serial Port          : Enabled
Suspend Mode              : Disabled        Parallel             : Disabled
HDD Power Down            : Disabled
Throttle Duty Cycle       : 62.5%
VGA Active Monitor        : Disabled
Resume By Ring            : Enabled
Resume By Alarm           : Disabled


Wake Up On LAN            : Enabled
CPU Clock Throttle        : Enabled         ESC  : Quit       ↑↓→←:Select Item
IRQ 8 Break Suspend                         F1   : Help       PU/PD/+/- : Modify
                                            F5   : Old Values (Shift)F2 : Color
                                            F6   : Load Failsafe Defaults
                                            F7   : Load Optimal Defaults
```

ACPI Function

ACPI is Advanced Configuration Power Management Interface. Selecting Enabled enables ACPI device node reporting from the BIOS to the operating system. This method is used by Windows NT and Windows 98 to power manage the computer. The default is Disabled.

Power Management

This option disables or sets the power management options. The options are:

- Max Saving   Maximum power savings. This option is only available for SL CPUs. Inactivity period is 1 minute in each mode.
- User Define   Set each mode individually. Select time-out periods in the PM timers section.
- Min Savings   Minimum power savings. Inactivity period is 1 hour in each mode (except the hard drive).
- Disabled   Turns off all power management features.

The default is Disabled.

PM Control by APM

Advanced Power Management (APM) provides better power savings. The default is No.

Video Off Method

Determines the manner in which the monitor is blanked. The options are:

• V/H SYNC+Blank	System turns off vertical and horizontal synchronization ports and writes blanks to the video buffer.
• DPMS Support	Select this option if your monitor supports the Display Power Management Signaling (DPMS) standard of the Video Electronics Standards Association (VESA). Use the software supplied for your video subsystem to select video power management values.
• Blank Screen	System only writes blanks to the video buffer.

The default setting is V/H SYNC+Blank.

Video Off After

As the system moves from lesser to greater power-saving modes, select the mode in which you want the monitor to blank. The default is Standby.

Modem Use IRQ

Name the interrupt request (IRQ) line assigned to the modem (if any) on your system. Activity of the selected IRQ awakens the system. The default setting is N/A.

Doze Mode

After the selected period of system inactivity (1 minute to 1 hour), the CPU clock runs at a slower speed while all other devices still operate at full speed. The default is Disabled.

Standby Mode

After the selected period of system inactivity (1 minute to 1 hour), the fixed disk drive and the video shut off while all other devices still operate at full speed. The default setting is Disabled.

Suspend Mode

After the selected period of system inactivity (1 minute to 1 hour), all devices except the CPU shut off. The default setting is Disabled.

HDD Power Down

After the selected period of drive inactivity (1 to 15 minutes), the hard disk drive powers down while all other devices remain active. The default is Disabled.

Throttle Duty Cycle

When the system enters Doze mode, the CPU clock runs only part of the time. You may select the percent of time that the clock runs. The default is 62.5%.

VGA Active Monitor

When Enabled, any video activity restarts the global timer for Standby mode. The default is Disabled.

Resume By Ring

When Enabled, an input signal on the serial Ring Indicator (RI) line (in other words, an incoming call on the modem) awakens the system from a soft off state. The default is Enabled.

Resume By Alarm

When Enabled, an alarm signal awakens the system from a soft off state. The default is Disabled.

Wake Up On LAN

When Enabled, a request from the LAN awakens the system from a soft off state. The default is Enabled.

CPU Clock Throttle

When Enabled this allows the power management routines to adjust the clock speed of the processor down to compensate for temperature. The default is Enabled.

IRQ 8 Break Suspend

You can Enable or Disable monitoring of IRQ8 (the Real Time Clock) so it does not awaken the system from Suspend mode. The default is Disabled.

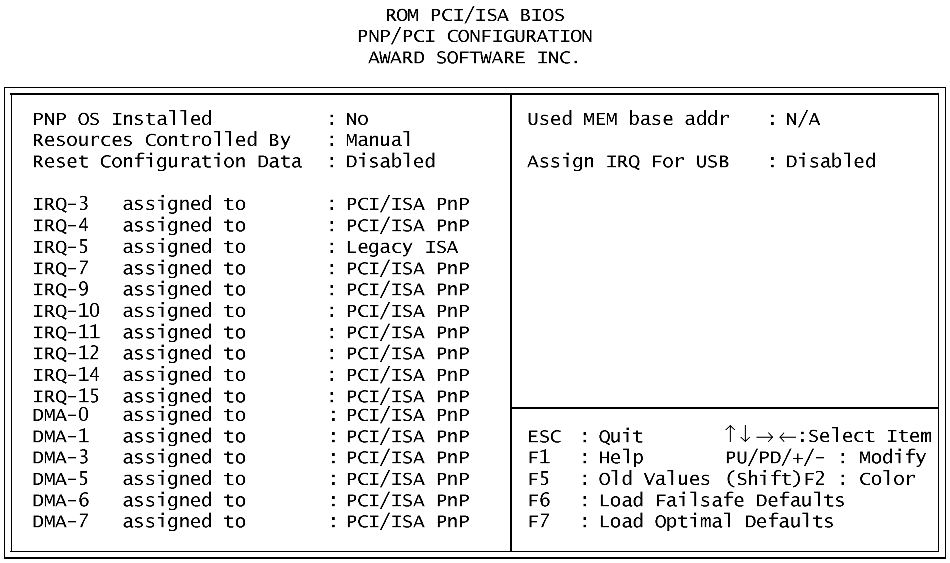
Reload Global Timer Events

When Enabled, an event occurring on each device listed below restarts the global timer for Standby mode.

- IRQ3-7, 9-15, NMI
- Primary IDE 0
- Primary IDE 1
- Secondary IDE 0
- Secondary IDE 1
- Floppy Disk
- Serial Port
- Parallel Port

PnP/PCI Configuration

This section describes the PNP / PCI options available.



```
                     ROM PCI/ISA BIOS
                   PNP/PCI CONFIGURATION
                    AWARD SOFTWARE INC.

PNP OS Installed          : No          Used MEM base addr    : N/A
Resources Controlled By   : Manual
Reset Configuration Data  : Disabled    Assign IRQ For USB    : Disabled

IRQ-3   assigned to       : PCI/ISA PnP
IRQ-4   assigned to       : PCI/ISA PnP
IRQ-5   assigned to       : Legacy ISA
IRQ-7   assigned to       : PCI/ISA PnP
IRQ-9   assigned to       : PCI/ISA PnP
IRQ-10  assigned to       : PCI/ISA PnP
IRQ-11  assigned to       : PCI/ISA PnP
IRQ-12  assigned to       : PCI/ISA PnP
IRQ-14  assigned to       : PCI/ISA PnP
IRQ-15  assigned to       : PCI/ISA PnP
DMA-0   assigned to       : PCI/ISA PnP
DMA-1   assigned to       : PCI/ISA PnP   ESC  : Quit       ↑↓→←:Select Item
DMA-3   assigned to       : PCI/ISA PnP   F1   : Help       PU/PD/+/- : Modify
DMA-5   assigned to       : PCI/ISA PnP   F5   : Old Values (Shift)F2 : Color
DMA-6   assigned to       : PCI/ISA PnP   F6   : Load Failsafe Defaults
DMA-7   assigned to       : PCI/ISA PnP   F7   : Load Optimal Defaults
```

PNP OS Installed

Select Yes if the system operating environment is Plug-and-Play aware (e.g., Windows 95). The default is No.

Resources Controlled By

The Plug-and-Play BIOS can automatically configure all the boot and Plug-and-Play compatible devices. If Auto is selected, all the interrupt request (IRQ) and DMA assignment and memory base address fields disappear, as the BIOS automatically assigns them. The default is Manual.

Reset Configuration Data

Select Enabled to reset Extended System Configuration Data (ESCD) when exiting Setup if a new add-on has been installed and the system reconfiguration has caused such a serious conflict that the operating system cannot boot. The default is Disabled.

IRQ *n* Assigned to

When resources are controlled manually, assign each system interrupt as one of the following types, depending on the type of device using the interrupt:

- Legacy ISA Devices compliant with the original PC AT bus specification, requiring a specific interrupt (such as IRQ4 for serial port 1).
- PCI/ISA PnP Devices compliant with the Plug-and-Play standard, whether designed for PCI or ISA bus architecture.

DMA *n* Assigned to

When resources are controlled manually, assign each system DMA channel as one of the following types, depending on the type of device using the interrupt:

- Legacy ISA Devices compliant with the original PC AT bus specification, requiring a specific DMA channel
- PCI/ISA PnP Devices compliant with the Plug-and-Play standard, whether designed for PCI or ISA bus architecture.

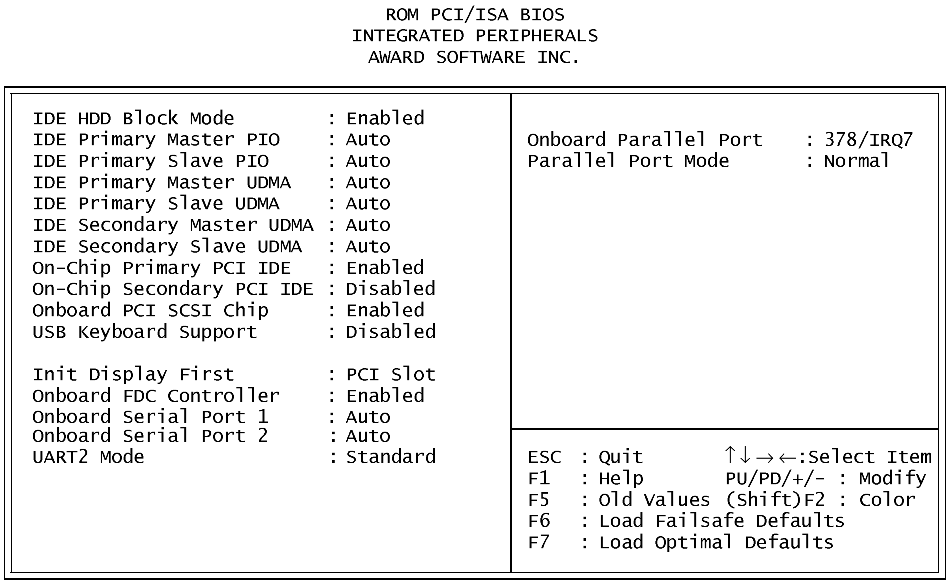
Used Mem Base Addr

Select a base address for the memory area used by any peripheral that requires high memory. The defaults are:

• Used MEM base addr	N/A
• Assign IRQ For USB	Disabled

Integrated Peripherals

This section describes the setup for integrated peripherals in the system.



```
                    ROM PCI/ISA BIOS
                 INTEGRATED PERIPHERALS
                  AWARD SOFTWARE INC.

 IDE HDD Block Mode        : Enabled
 IDE Primary Master PIO    : Auto       Onboard Parallel Port    : 378/IRQ7
 IDE Primary Slave PIO     : Auto       Parallel Port Mode       : Normal
 IDE Primary Master UDMA   : Auto
 IDE Primary Slave UDMA    : Auto
 IDE Secondary Master UDMA : Auto
 IDE Secondary Slave UDMA  : Auto
 On-Chip Primary PCI IDE   : Enabled
 On-Chip Secondary PCI IDE : Disabled
 Onboard PCI SCSI Chip     : Enabled
 USB Keyboard Support      : Disabled

 Init Display First        : PCI Slot
 Onboard FDC Controller    : Enabled
 Onboard Serial Port 1     : Auto
 Onboard Serial Port 2     : Auto       ESC  : Quit       ↑↓→←:Select Item
 UART2 Mode                : Standard   F1   : Help       PU/PD/+/- : Modify
                                        F5   : Old Values (Shift)F2 : Color
                                        F6   : Load Failsafe Defaults
                                        F7   : Load Optimal Defaults
```

IDE HDD Block Mode

Block mode is also called block transfer, multiple commands, or multiple sector read / write. If the IDE hard drive supports block mode, select Enabled for automatic detection of the optimal number of block read / writes per sector the drive can support. The default is Enabled.

IDE Primary Master PIO

This IDE PIO (Programmed Input / Output) field allows setting a PIO mode (0-4) for the IDE devices that the onboard IDE interface supports. Modes 0 through 4 provide successively increased performance. In Auto mode, the system automatically determines the best mode for each device. The default is Auto.

IDE Primary Slave PIO

This IDE PIO (Programmed Input / Output) field allows setting a PIO mode (0-4) for the IDE devices that the onboard IDE interface supports. Modes 0 through 4 provide successively increased performance. In Auto mode, the system automatically determines the best mode for each device. The default is Auto.

IDE Primary/Secondary Master UDMA

Ultra DMA / 33 implementation is possible only if the IDE hard drive supports it and the operating environment includes a DMA driver (Windows 95 OSR2 or a third-party IDE bus master driver). If the hard drive and the operating system both support Ultra DMA / 33, select Auto to enable BIOS support. The default is Auto.

IDE Primary/Secondary Slave UDMA

Ultra DMA / 33 implementation is possible only if the IDE hard drive supports it and the operating system includes a DMA driver (Windows 95 OSR2 or a third-party IDE bus master driver). If the hard drive and the operating system both support Ultra DMA / 33, select Auto to enable BIOS support. The default is Auto.

On-Chip Primary PCI IDE

The integrated peripheral controller contains an IDE interface with support for two IDE channels. Select Enabled to activate each channel separately. The default is Enabled.

On-Chip Secondary PCI IDE

The integrated peripheral controller contains an IDE interface with support for two IDE channels. Select Enabled to activate each channel separately. When this option is disabled, the IDE Primary Master PIO and IDE Primary Slave PIO options in this screen are removed. The default is Disabled.

Onboard PCI SCSI Chip

Select Enabled if your system contains a built-in PCI SCSI controller. The default is Enabled.

USB Keyboard Support

Select Enabled if the operating system contains a Universal Serial Bus (USB) controller and you have a USB keyboard. The default is Disabled.

Init Display First

Initialize the AGP video display before initializing any other display device on the system. Thus the AGP display becomes the primary display. The default is PCI Slot.

Onboard FDC Controller

Select Enabled to activate the floppy disk controller (FDC) installed on the system board. If an add-in FDC was installed or the system has no floppy drive, select Disabled in this field. The default is Enabled.

Onboard Serial Port 1/2

Select an address and corresponding interrupt for the first and second serial ports. The default for both ports is Auto.

UART 2 Mode

Select an operating mode for the second serial port. The infrared options listed below are not supported by the VMIVME-7696. The options are:

• Standard	RS-232C serial port
• IrDA 1.0	N / A
• MIR 0.57M	N / A
• MIR 1.15M	N / A
• FIR	N / A
• ASK IR	N / A

The default for this setting is Standard.

Onboard Parallel Port

Select an address and corresponding interrupt for the physical parallel (printer) port. The default setting is 378/IRQ7.

Parallel Port Mode

Select an operating mode for the onboard parallel (printer) port. The default setting is Normal.

Appendix

D

LANWorks BIOS

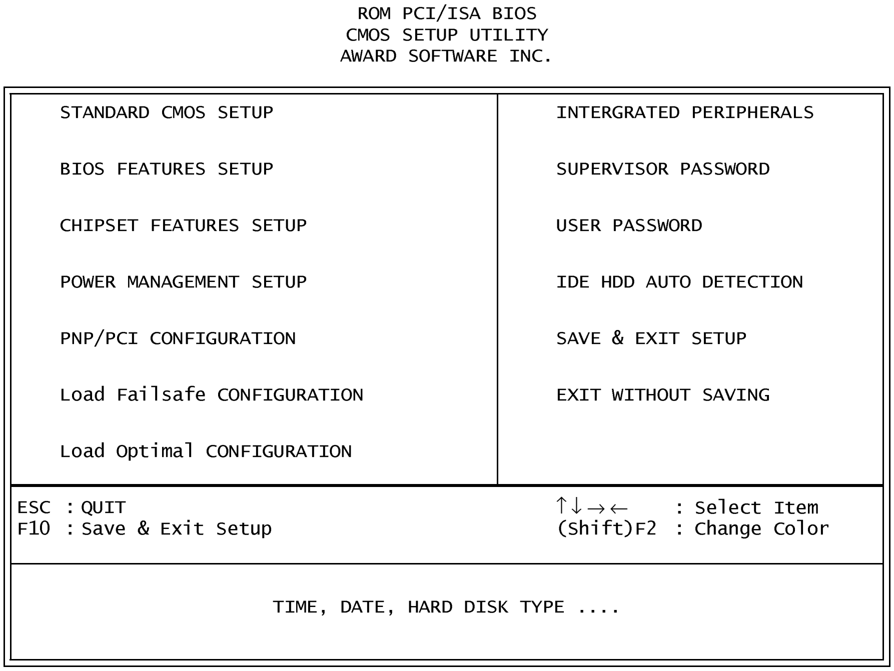
Contents

Introduction

The VMIVME-7696 includes the LANWorks option which allows the VMIVME-7696 to be booted from a network. This appendix describes the LANWorks BIOS Setup screen, and the procedures to enable this option.

System BIOS Setup Utility

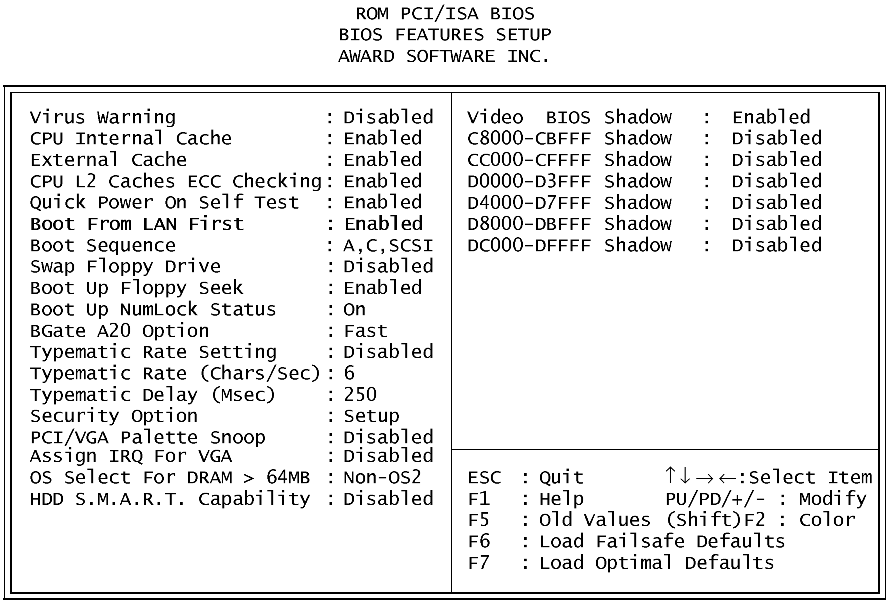
To enable the LANWorks BIOS option reboot the VMIVME-7696 and when prompted, press the Delete key to access the System BIOS Setup Utility screen shown below.



```
                        ROM PCI/ISA BIOS
                       CMOS SETUP UTILITY
                      AWARD SOFTWARE INC.

    STANDARD CMOS SETUP                     INTERGRATED PERIPHERALS

    BIOS FEATURES SETUP                     SUPERVISOR PASSWORD

    CHIPSET FEATURES SETUP                  USER PASSWORD

    POWER MANAGEMENT SETUP                  IDE HDD AUTO DETECTION

    PNP/PCI CONFIGURATION                   SAVE & EXIT SETUP

    Load Failsafe CONFIGURATION             EXIT WITHOUT SAVING

    Load Optimal CONFIGURATION

ESC  : QUIT                                 ↑↓→←        : Select Item
F10  : Save & Exit Setup                    (Shift)F2  : Change Color

                   TIME, DATE, HARD DISK TYPE ....
```

Using the arrow keys, highlight *BIOS Features Setup*, and press the Return key. This will display the BIOS Features Setup screen.

BIOS Features Setup



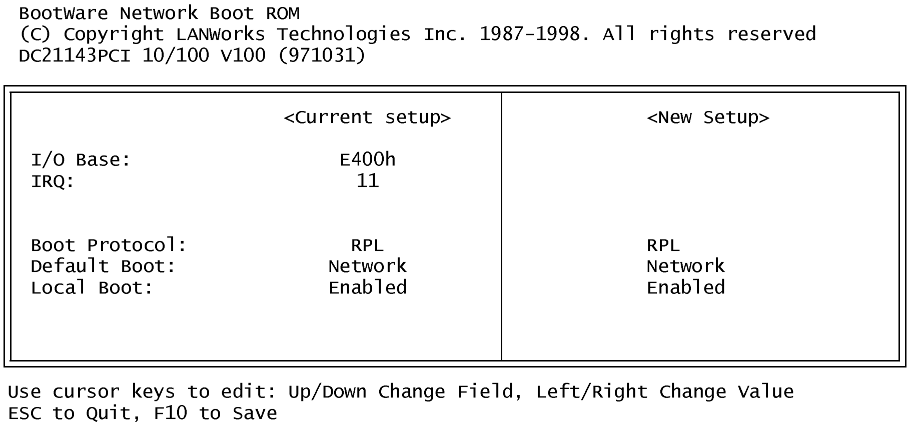
```
                         ROM PCI/ISA BIOS
                        BIOS FEATURES SETUP
                        AWARD SOFTWARE INC.

Virus Warning             : Disabled   Video  BIOS Shadow   :  Enabled
CPU Internal Cache        : Enabled    C8000-CBFFF Shadow   :  Disabled
External Cache            : Enabled    CC000-CFFFF Shadow   :  Disabled
CPU L2 Caches ECC Checking: Enabled    D0000-D3FFF Shadow   :  Disabled
Quick Power On Self Test  : Enabled    D4000-D7FFF Shadow   :  Disabled
Boot From LAN First       : Enabled    D8000-DBFFF Shadow   :  Disabled
Boot Sequence             : A,C,SCSI   DC000-DFFFF Shadow   :  Disabled
Swap Floppy Drive         : Disabled
Boot Up Floppy Seek       : Enabled
Boot Up NumLock Status    : On
BGate A20 Option          : Fast
Typematic Rate Setting    : Disabled
Typematic Rate (Chars/Sec): 6
Typematic Delay (Msec)    : 250
Security Option           : Setup
PCI/VGA Palette Snoop     : Disabled
Assign IRQ For VGA        : Disabled   ESC  : Quit       ↑↓→←:Select Item
OS Select For DRAM > 64MB : Non-OS2    F1   : Help       PU/PD/+/- : Modify
HDD S.M.A.R.T. Capability : Disabled   F5   : Old Values (Shift)F2 : Color
                                       F6   : Load Failsafe Defaults
                                       F7   : Load Optimal Defaults
```

Using the arrow keys, enable *Boot From LAN First*. Exit the BIOS setup, saving changes.

When prompted, press “Control-Alt-B” as the VMIVME-7696 reboots. This will activate the LANWorks BIOS setup screen.

LANWorks BIOS Setup

Below is the screen in which the various options for booting through LANWorks are set.



```
BootWare Network Boot ROM
(C) Copyright LANWorks Technologies Inc. 1987-1998. All rights reserved
DC21143PCI 10/100 V100 (971031)

                       <Current setup>                <New Setup>

 I/O Base:                  E400h
 IRQ:                        11


 Boot Protocol:              RPL                       RPL
 Default Boot:             Network                     Network
 Local Boot:               Enabled                     Enabled

Use cursor keys to edit: Up/Down Change Field, Left/Right Change Value
ESC to Quit, F10 to Save
```

Boot Protocol

The Boot Protocol required is dependent on the network system being utilized, check with your network administrator for the correct protocol. The following options are available in the Boot Protocol field:

- **RPL** **(Default)**
- TCP/IP BootP
- TCP/IP DHCP
- Netware 802.3
- Netware 802.2
- Netware EthII

Default Boot

The following options are available in the Default Boot field:

- **Network (Default) -** Selects Network Boot as the default boot device
- Local **-** Selects a Local Drive as the default boot device

Local Boot

The following options are available in the Local Boot field:

- Disabled **-** Disables Local Boot from a Hard Drive or Floppy
- **Enabled** (Default) **-** Enables Local Boot from a Hard Drive or Floppy

Appendix

E

SCSI Selection Utility

Contents

Introduction

The SCSI BIOS includes a configuration utility that enables users to change the VMIVME-7696 SCSI adapter settings. The utility lets users list the SCSI IDs of devices on the host adapter, format SCSI disk drives, and check drives for defects. This section describes the utility default and permitted settings, and the procedure for using the utility.

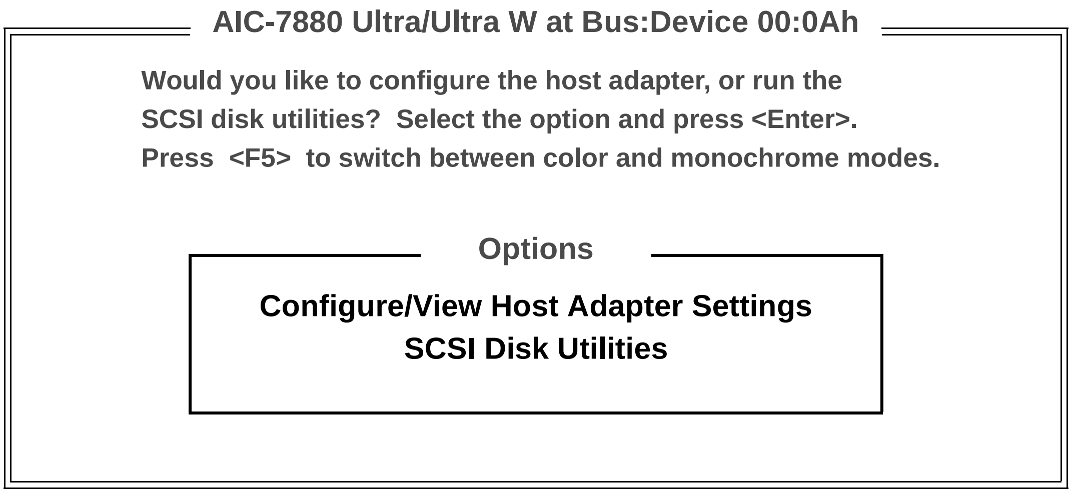
In this appendix, the VMIVME-7696 SCSI adapter will be referred to as the ‘host adapter’ or as the ‘adapter.’

During boot-time, the Power-On Self Test (POST) software displays the message:

```
<<< Press <Ctrl><A> for Select SCSI (TM) Utility! >>>
```

Initiating this key sequence presents to the operator the SCSI Selection Utility main menu which can be used to edit the Host Adapter Settings or the SCSI Disk Utility.

Options available to the operator are designated with dark text in the following graphics. The parameters included are default values.



AIC-7880 Ultra/Ultra W at Bus:Device 00:0Ah

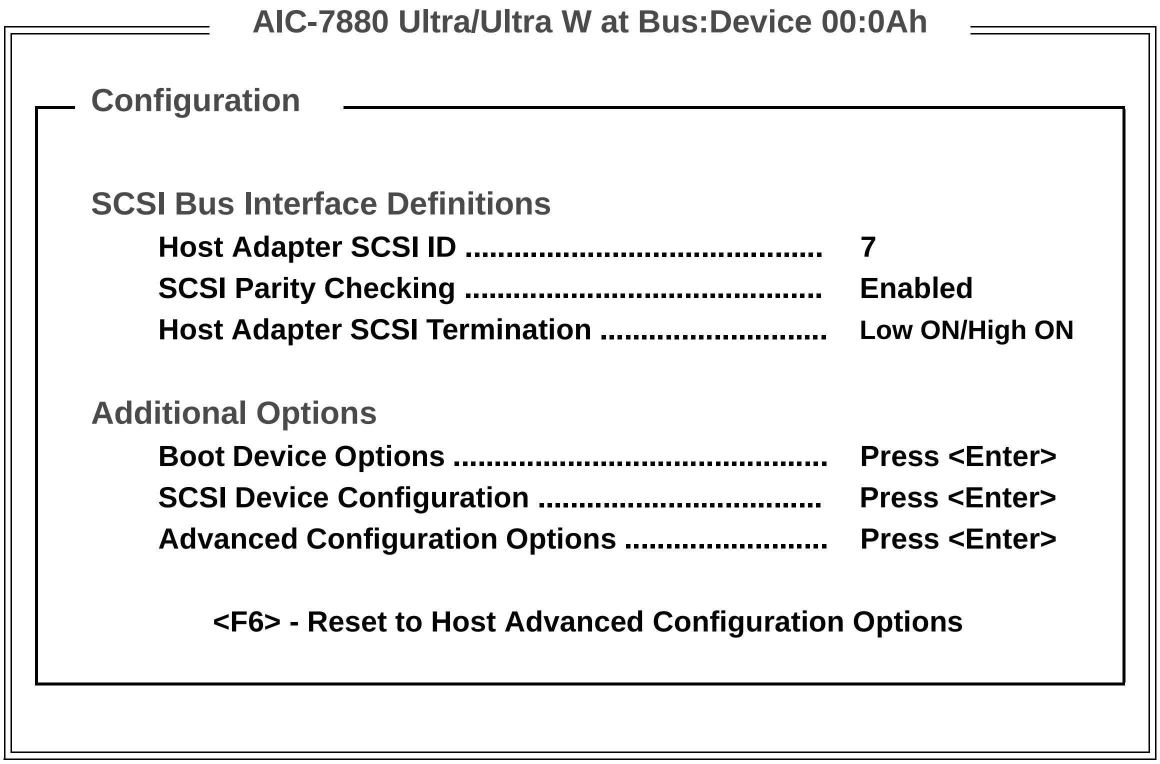
Would you like to configure the host adapter, or run the
SCSI disk utilities? Select the option and press <Enter>.
Press <F5> to switch between color and monochrome modes.

Options

Configure/View Host Adapter Settings
SCSI Disk Utilities

Configure/View Host Adapter Settings

Selection of the first main menu item, the **Configure/View Host Adapter Settings,** allows the user to adjust the SCSI Bus Interface Definitions.



AIC-7880 Ultra/Ultra W at Bus:Device 00:0Ah

Configuration

SCSI Bus Interface Definitions

Host Adapter SCSI ID 7

SCSI Parity Checking .... Enabled

Host Adapter SCSI Termination .... Low ON/High ON

Additional Options

Boot Device Options .... Press <Enter>

SCSI Device Configuration .... Press <Enter>

Advanced Configuration Options .... Press <Enter>

<F6> - Reset to Host Advanced Configuration Options

SCSI Bus Interface Definitions

The SCSI bus Interface Definition default settings configure the unit for SCSI bus operation. The settings described here are for a host adapter SCSI ID, host adapter termination, and parity checking.

Host Adapter SCSI ID

Each device on the SCSI bus must have a unique SCSI ID. Allowable IDs include 0 through 7 on 8-bit adapters, and 0 through 15 on 16-bit adapters. The ID uniquely defines each SCSI device on the bus. The ID also determines which device controls the bus when two or more devices try to use the bus at the same time. For 8-bit devices, ID 7 has the highest priority and ID 0 has the lowest priority. For 16-bit devices, the priority of IDs is 7-0, then 15-8: in this case, ID 7 has the highest priority and ID 8 has the lowest priority.

The Adapter has the highest priority on the SCSI bus, since each adapter (8- or 16-bit) is defaulted to SCSI ID of 7. Options range from 0 through 15.

SCSI Parity Checking

SCSI parity checking is used to verify the accuracy of data transfer on the SCSI bus for each adapter. If a device on the SCSI bus does not support SCSI parity, then disable the parity checking. Most SCSI devices support SCSI parity. The default setting is Enabled.

Host Adapter SCSI Termination

A set of resistors, called Terminators, must be either installed in or enabled on the first and last SCSI devices on each SCSI bus. Without termination, the devices will not operate properly. The adapter is usually the SCSI device at one end of the bus and will cause the termination to be enabled by default. On default, termination is enabled for the low byte (bits 0-7) and high byte (bits 8-15) of the SCSI bus. Refer to the following to set termination:

- Low ON / High ON (the default) - Adapter is at end of SCSI bus; the bus has only 8-bit devices or only 16-bit devices.
- Low ON / High ON - Adapter is at end of SCSI bus; the bus has 8-bit and 16-bit SCSI devices. Last device must be 16-bit and terminated.
- Low OFF / High OFF - Adapter is not at end of SCSI bus; the bus has only 16-bit SCSI devices.
- Low OFF / High ON - Adapter is not at end of SCSI bus; the bus has both 8-bit and 16-bit SCSI devices

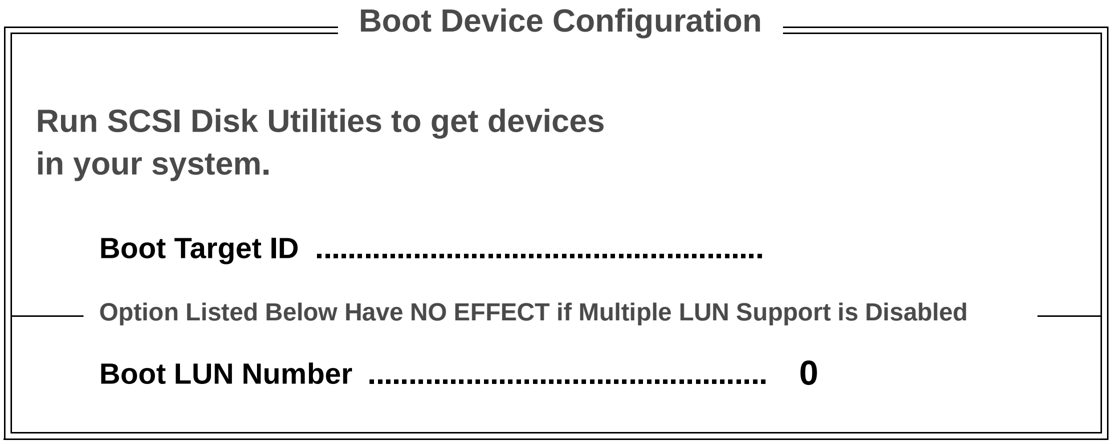
Additional Options

Boot Device Options

With the Boot Device options the user specifies the boot device. The default boot device is SCSI ID 0 and logical unit number (LUN) 0. To specify a different boot device, choose a different SCSI ID: ID 0 through 7 on 8-bit adapters, or ID 0 through 15 on 16-bit adapters. The Boot Target ID default is 0.

The operator must also specify the boot logical unit number (LUN). If the boot device has multiple logical, units this LUN can be 0 through 7 (on 8-bit or 16-bit adapters).

The Boot LUN Number default is 0.



Boot Device Configuration

Run SCSI Disk Utilities to get devices in your system.

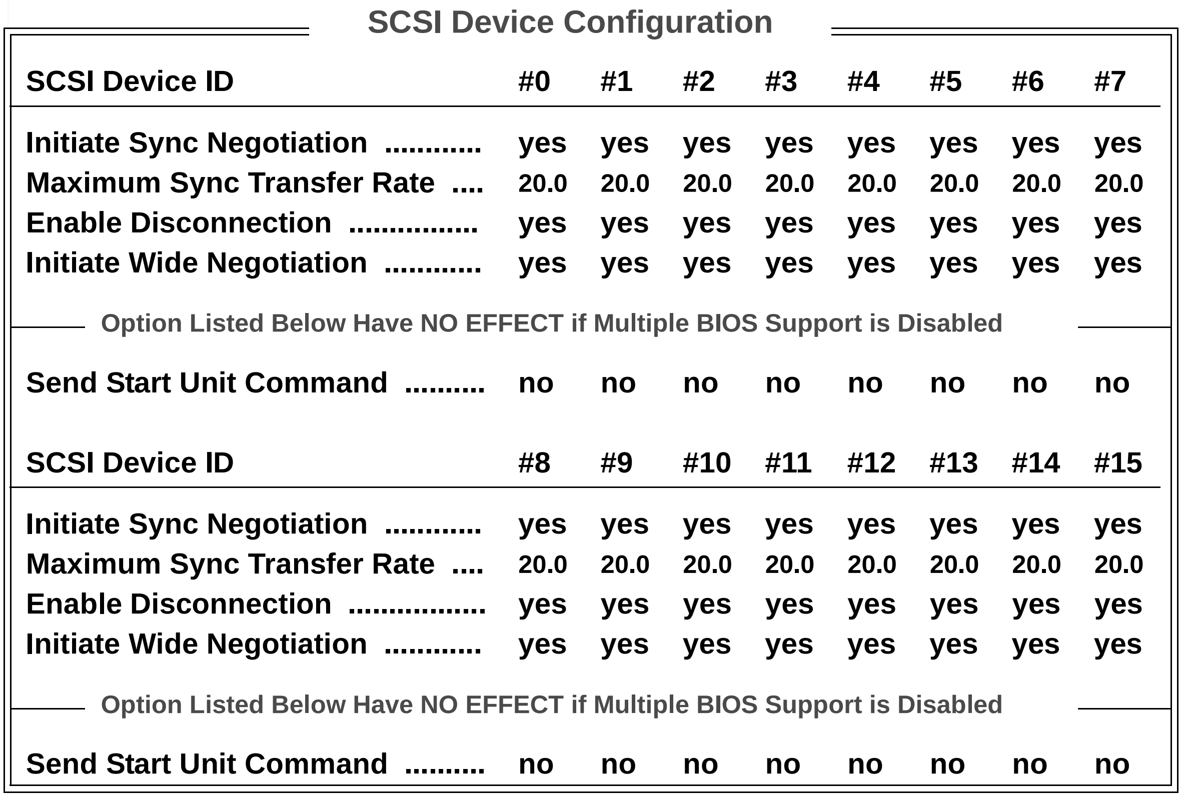
Boot Target ID ....

Option Listed Below Have NO EFFECT if Multiple LUN Support is Disabled

Boot LUN Number **0**

SCSI Device Configuration

Each device on the SCSI bus requires configuration parameters that must be defined prior to device operation. These configuration parameters define how devices will transfer command and data information on the bus. The SCSI Device Configuration default options are shown in the menu illustration shown below.



SCSI Device Configuration

SCSI Device ID	#0	#1	#2	#3	#4	#5	#6	#7
Initiate Sync Negotiation	yes	yes	yes	yes	yes	yes	yes	yes
Maximum Sync Transfer Rate	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
Enable Disconnection	yes	yes	yes	yes	yes	yes	yes	yes
Initiate Wide Negotiation	yes	yes	yes	yes	yes	yes	yes	yes

Option Listed Below Have NO EFFECT if Multiple BIOS Support is Disabled

Send Start Unit Command	no	no	no	no	no	no	no	no

SCSI Device ID	#8	#9	#10	#11	#12	#13	#14	#15
Initiate Sync Negotiation	yes	yes	yes	yes	yes	yes	yes	yes
Maximum Sync Transfer Rate	20.0	20.0	20.0	20.0	20.0	20.0	20.0	20.0
Enable Disconnection	yes	yes	yes	yes	yes	yes	yes	yes
Initiate Wide Negotiation	yes	yes	yes	yes	yes	yes	yes	yes

Option Listed Below Have NO EFFECT if Multiple BIOS Support is Disabled

Send Start Unit Command	no	no	no	no	no	no	no	no

Initiate Sync Negotiation

Setting the Synchronous Negotiation to yes allows the SCSI adapter and its attached SCSI devices to transfer data in synchronous mode. Such transfer is faster than asynchronous data transfer.

The default setting is **Yes** causing the adapter to initiate synchronous negotiation with the SCSI device. Setting the value to *No* causes the adapter to not initiate synchronous negotiation; however, the adapter always responds to synchronous negotiate if the SCSI device initiates it. If neither the adapter nor the SCSI device negotiates for synchronous data transfer, data is transferred in asynchronous mode.

The majority of SCSI devices support synchronous negotiation. If a device does not support synchronous negotiation, then the adapter automatically transfers the data in asynchronous mode.

Maximum Synchronous Transfer Rate

The maximum synchronous transfer rate that the adapter will negotiate is defined by this setting. The default rates of the transfer are defined in the above SCSI Device Configuration menu. The adapter automatically negotiates for the rate requested by the device. The default setting is 20.0 for all devices.

If the adapter is set to *not* negotiate for the synchronous data transfer, then the value selected is the maximum rate at which the adapter accepts data from the device during negotiation. This is standard SCSI protocol.

Enable Disconnection

This menu item defines whether the SCSI device may disconnect from the SCSI bus. The disconnect / reconnect function allows the adapter to perform other operations on the SCSI bus while the SCSI device is temporarily disconnected. The default setting is Yes, meaning that the SCSI device may disconnect.

The device may contain the logic to decide not to disconnect even if it is permitted by the adapter setting. This can be configured on the SCSI device. The SCSI device cannot disconnect from the SCSI bus when the Enable Disconnect is set to *No*.

The Enable Disconnection should be set to *yes* if the adapter connects to two or more SCSI devices. This will optimize SCSI bus performance. If the adapter connects to only one SCSI device, set the item to *No* to achieve better performance.

Initiate Wide Negotiation

Leave this option set to *yes*. The adapter will not attempt wide negotiation with 8-bit devices even if the bus is set to wide. This allows the adapter to initiate wide negotiation with 16-bit devices. The default setting is Yes.

Send Start Unit Command

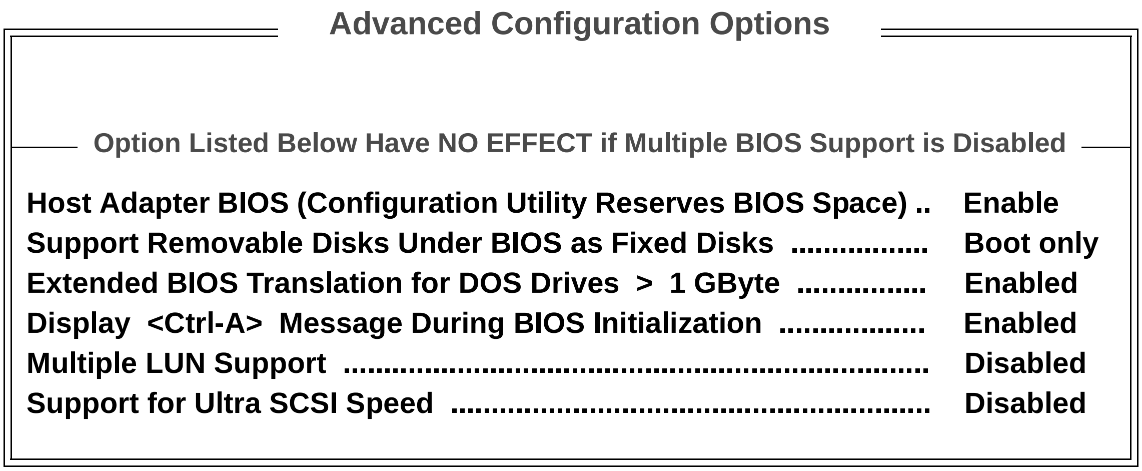
This function serves to reduce the load on the computer's power supply. Enabling the option allows the adapter to turn on SCSI devices sequentially during boot-up. Specifically, the setting defines whether the adapter sends the Start Unit command to the SCSI device. This is the SCSI command 1B. The default settings is No.

The SCSI BIOS must be enabled before you can enable this option for the device.

If you enable the Send Start Unit Command for more than one SCSI device, the adapter sends the Start Unit command to the boot device as defined in the Boot Device Configuration menu (page 1-141). After this device responds, Start Unit commands are sent to the additional devices, beginning with the device which has the lowest SCSI ID.

Advanced Configuration Options

The SCSI BIOS includes advanced configuration setting. Users are discouraged from changing these options. The default settings are included in the following menu.



```
                    Advanced Configuration Options

   Option Listed Below Have NO EFFECT if Multiple BIOS Support is Disabled

Host Adapter BIOS (Configuration Utility Reserves BIOS Space) ..   Enable
Support Removable Disks Under BIOS as Fixed Disks ................  Boot only
Extended BIOS Translation for DOS Drives > 1 GByte ...............  Enabled
Display <Ctrl-A> Message During BIOS Initialization ..............  Enabled
Multiple LUN Support ..............................................  Disabled
Support for Ultra SCSI Speed ......................................  Disabled
```

Host Adapter BIOS

This setting enables or disables the SCSI BIOS. The BIOS must be enabled if you want the computer to boot from a SCSI hard disk drive connected to the adapter. The default setting is Enabled. All options on the Advanced Configuration Option menu are disabled with this option.

Note

If the devices on the SCSI bus are controlled by device drivers then a BIOS for these is not necessary. The SCSI BIOS can be disabled to free up about 16 Kbyte of memory. This will also reduce boot time by 60 seconds.

Support Removable Disks Under BIOS as Fixed Disks

This option provides control of removable, media drives. The default is **Boot Only**. Selection options include:

Boot Only - The removable, media drive designated as the boot device is considered as a hard disk drive.

All Disks - The removable, media drives supported by the SCSI BIOS are considered as hard disk drives. There is no effect on drives under NetWare which automatically supports removable, media drives as fixed drives.

Disabled -No removable, media drives running under DOS are considered as hard disk drives. Driver software is necessary under these circumstances since the drives are not controlled by the BIOS.

Extended BIOS Translation for DOS Drives > 1 Gbyte

The SCSI BIOS includes an extended translation scheme that supports drives up to 8 Gbyte under MS-DOS. With this option enabled, drives managed by the BIOS use the extended translation if their formatted capacity is greater than 1 Gbyte. Standard translation is used on drives smaller than 1 Gbyte. The default is Enabled.

Back up disk drives before changing the translation scheme. All data is erased upon changing from one scheme to another.

Use the MS-DOS `fdisk` utility as normal to partition a disk larger than 1 Gbyte. The cylinder size increases to 8 Mbyte under extended translation; therefore, the partition size must be a multiple of 8 Mbyte. Requesting a size that is not a multiple of 8 Mbyte causes `fdisk` to round up to the nearest whole multiple of 8 Mbyte.

Display <Ctrl-A> Message During BIOS Initialization

The option controls the display of the boot prompt message:

Press <Ctrl><A> for SCSI Select (TM) Utility!

Designating the option as Enable causes the prompt to display during every computer boot. The default option is Enabled. The default option hot key sequence is <Ctrl-A>. This can be altered by editing the `absegs.inc` file.

Multiple LUN Support

This option allows the BIOS to support multiple logical units for any device. The default entry is Disabled.

Support For Ultra SCSI Speed

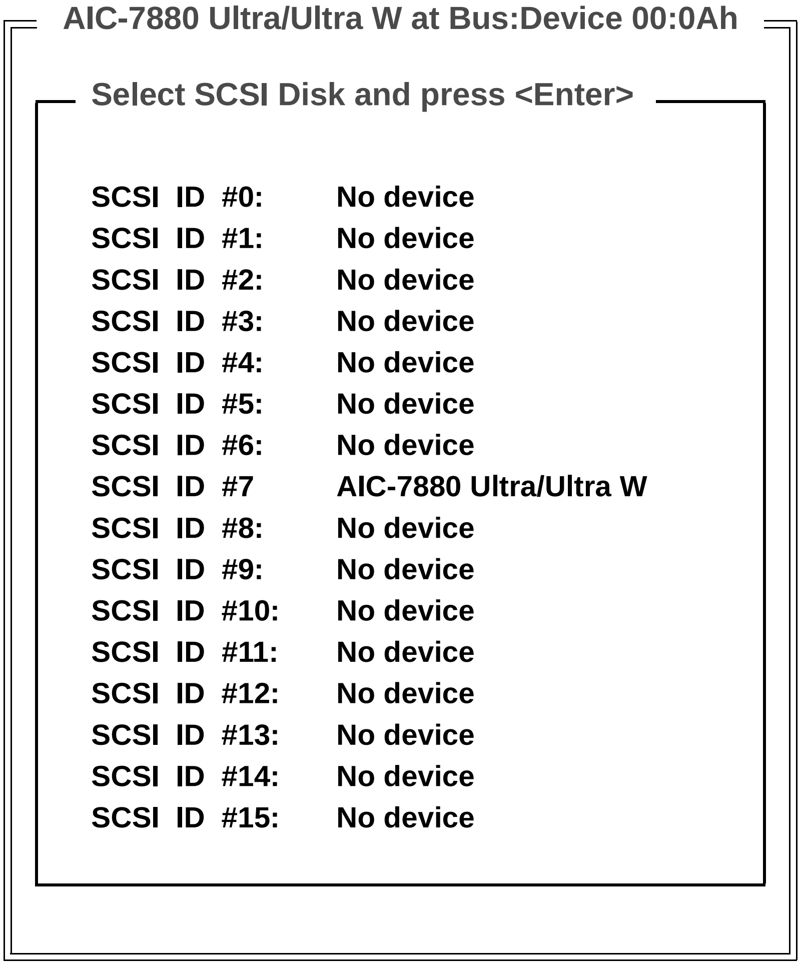
This option appears only if the BIOS is configured to include Ultra SCSI support. To support Ultra SCSI speeds, the option must be Enabled, and the Jumper J14 on the VMIVME-7696 CPU board must be open. The default entry is Disabled and the Jumper J14 is installed.

SCSI Disk Utilities

The SCSI Disk Utilities allow the users to perform disk setup and configuration operations. These operations include listing SCSI IDs of the devices on the host adapter, formatting SCSI disk drives, and checking drives for defects.

Select SCSI Disk

To list and select SCSI ID’s of devices, select SCSI Disk Utilities from the main menu. This allows the user to view the list of current SCSI devices and their SCSI ID Number and insure that no duplicates are present.



```
AIC-7880 Ultra/Ultra W at Bus:Device 00:0Ah

Select SCSI Disk and press <Enter>

SCSI  ID  #0:      No device
SCSI  ID  #1:      No device
SCSI  ID  #2:      No device
SCSI  ID  #3:      No device
SCSI  ID  #4:      No device
SCSI  ID  #5:      No device
SCSI  ID  #6:      No device
SCSI  ID  #7       AIC-7880 Ultra/Ultra W
SCSI  ID  #8:      No device
SCSI  ID  #9:      No device
SCSI  ID  #10:     No device
SCSI  ID  #11:     No device
SCSI  ID  #12:     No device
SCSI  ID  #13:     No device
SCSI  ID  #14:     No device
SCSI  ID  #15:     No device
```

Selection of a device allows the operator to format or verify the disk on the device.

Formatting a Disk

Most SCSI drives are preformatted; however, if formatting is necessary, the operator can use the SCSI BIOS to perform a low-level format on the drive. The formatting is compatible with most SCSI disk drives. The operating system's partitioning and high-level formatting utilities, such as MS-DOS `fdisk` and `format`, require that a disk have a low-level format. To format a disk:

- Select the device from the list of SCSI devices that you want to format.
- From the submenu that appears, select Format disk.
- At the confirmation prompt, select Yes to format, or No to cancel.

Note

You cannot abort formatting once it has started.

Verifying a Disk

Using the Verify Disk option, the operator can scan a device for defects. If bad blocks are found by the utility, the operator is prompted to reassign them so they are not further used. To conduct a verify:

- Select the device from the list of SCSI devices that you want to Verify.
- From the submenu that appears, select Verify disk.
- At the confirmation prompt, select Yes to reassign the bad blocks, or No to not alter and leave as is.
- To abort the verification, press Esc at any time.

Appendix

F

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-7696. The CPU board on the VMIVME-7696 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-7696 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-7696 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 could take over control of the board and proceed with a custom configuration and/or custom application. A user application could override the configuration set by the BIOS and reconfigure the system or it could accept what the BIOS initialized.

BIOS Control Overview

There are two areas on the VMIVME-7696 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-7696 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-7696.

Functional Overview

The block diagram included in Figure F-1 on page 151 illustrates the VMIVME-7696 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.

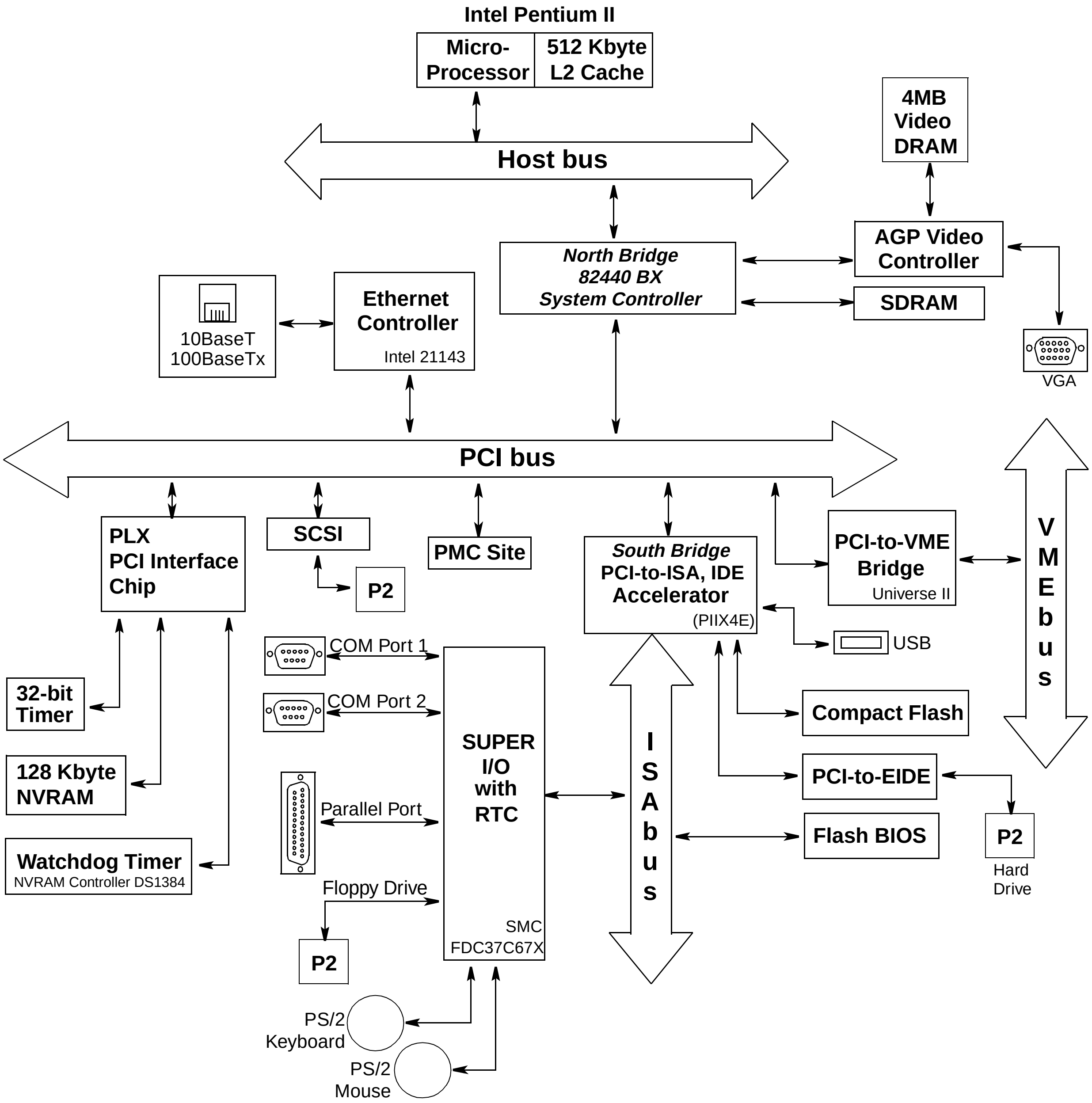


Figure F-1 VMIVME-7696 Block Diagram

Data Book References

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

2. Intel 21143 10/100 Mb/s Ethernet LAN Controller
   Intel
   www.intel.com

3. VMIVME-7696 User Manual
   500-007696-000 Product Manual
   500-007696-001 VMIVME-7696, Tundra Universe II-Based VMEbus
   Interface Option Product Manual

4. 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)

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

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

7. Flash ChipSet Product Manual
   SanDisk Corporation
   140 Caspian Court
   Sunnyvale, CA 94089-9820

8. 82C54 CHMOS Programmable Internal Timer
   Intel Corporation
   2200 Mission College Boulevard
   P.O. Box 58119
   Santa Clara, CA 95052-8119

9. DS 1284 Watchdog Timekeeping Controller
   Dallas Semiconductor
   4461 South Beltwood Pwky
   Dallas, TX 75244-3292

10. Intel 440Bx AGP Set: 82443 Bx Host Bridge Controller
    April 1998, Order Number 290633-001
    Intel Corp.
    P.O. Box 58119
    Santa Clara CA 95052-8119
    (408) 765-8080
    www.intel.com

11. Adaptec A-IC-7880 Ultra / FAST SCSI Host Adapter
    Adaptec Inc.
    691 South Milpitas Blvd.
    Milpitas CA 95035

12. S3 Trio 3d AGP Video Controller
    P.O. Box 58058
    Santa Clara CA 95052-8058
    (408) 588-8000

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-7696 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-7696 are configured by the BIOS at boot-up and fall under the realm of 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 F-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-7696 however, the Dynamic Random Access Memory (DRAM), Battery-Backed SRAM, Timers, 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 F-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
Parallel Port	N/A	[0x100 - 0xFFC] [0x100 - 0xFF8]	IRQ1 - IRQ15	4 8	$378
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	
System COMM Register	Nonrelocatable 0xD800E	N/A		2	

Table F-1 ISA Device Mapping Configuration (Continued)

Device	Memory Space	I/O Address Space	PIC Interrupt Options	Byte Address Boundary	Default
VMEBERR Address Register	Nonrelocatable 0xD8010	N/A		4	
VMEBERR Address Modifier Register	Nonrelocatable 0xD8014	N/A	N/A	2	
Board ID Register	Nonrelocatable 0xD8016	N/A	N/A	2	

PCI Devices

PCI devices are fully configured under I/O and/or Memory address space. Table F-3 describes the PCI bus devices that are on-board the VMIVME-7696 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 F-2 PCI Device Mapping Configuration

Device	Memory Space	I/O Address Space	PIC Interrupt Options
S3 AGP Video	Anywhere	N/A	N/A
Universe II* PCI-to-VMEbus Bridge	Anywhere	N/A	PCI Defined
Ethernet	Anywhere	Anywhere	PCI Defined
Timer Registers	Anywhere	N/A	IRQ5 Fixed
NVSRAM Registers	Anywhere	N/A	PCI Defined
Watchdog Registers	Anywhere	N/A	PCI Defined
PMC Site	Anywhere	Anywhere	PCI Defined
SCSI	Anywhere	Anywhere	PCI Defined

** Refer to the VMIVME-7696-001 User's Manual.*

Device Interrupt Definition

PC/AT Interrupt Definition

The interrupt hardware implementation on the VMIVME-7696 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 F-2 on page 157 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-7696 BIOS maps the IRQx lines to the appropriate device per the standard ISA architecture. Reference Figure F-2 on page 157. This initialization operation cannot be changed; however, a custom application could reroute the interrupt configuration after the BIOS has completed the initial configuration cycle.

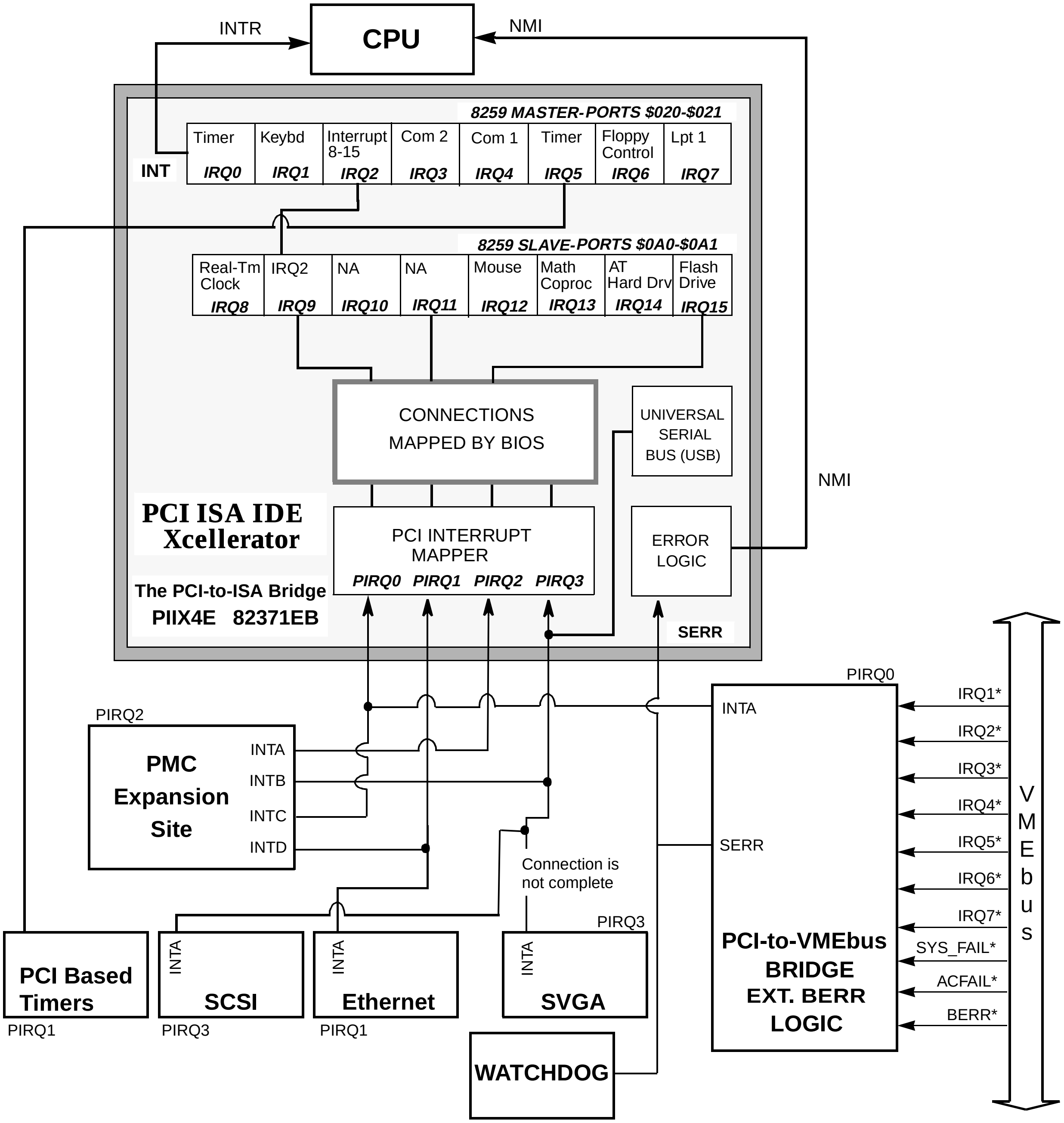


Figure F-2 BIOS Default Connections for the PC Interrupt Logic Controller

PCI Device Interrupt Map

The PCI bus-based external devices include the PCI expansion site, 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 PIIX4. This mapping is illustrated in Figure F-2 on page 157 and is defined in Table F-3.

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

Table F-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	3	1
Battery Backed SRAM	PLX 9052	0x114A	7696	AD25	N/A	N/A		N/A
Timers	PLX 9052	0x114A	7696	AD25	N/A	IRQ5 (ISA)	8	N/A
Watchdog Timer	PLX 9052	0x114A	7696	AD25	N/A	N/A	9	N/A
SCSI	Adaptec 7880	0x9004	0x8078	AD29	INTA	PIRQ3	11	N/A
AGP Video	S3 Trio 3d	0x5333	0x8904	N/A	N/A	N/A	12	N/A
PMC Expansion Site	N/A	Board Specific	Board Specific	AD31	INTA	PIRQ2	N/A	N/A
	N/A	Board Specific	Board Specific	AD 31	INTB	PIRQ3	N/A	N/A
	N/A	Board Specific	Board Specific	AD 31	INTC	PIRQ0	N/A	N/A
	N/A	Board Specific	Board Specific	AD 31	INTD	PIRQ1	N/A	N/A
Power Management	PIIX4 82371EB Function 03	0x8086	0x7113	AD18	N/A	N/A	1	N/A
PCI-to-ISA Bridge	PIIX4 82371EB Function 00	0x8086	0x7110	AD18	N/A	N/A	1	N/A
Ethernet Controller	DEC 21143	0x1011	0x0019	AD22	INTA	PIRQ1	2	N/A
PCI IDE Controller	PIIX4 82371EB Function 01	0x8086	0x7111	AD18	N/A	N/A	1	N/A
Universal Serial Bus (USB)*	PIIX4 82371EB Function 02	0x8086	0x7112	AD18	INTD	PIRQ3	1	N/A
PCI Host Bridge	Intel 440 BX	0x8086	0x7190	N/A	N/A	N/A	10	N/A
PCI-PCI Bridge for AGP Interface	Intel 440BX	0x8086	0x7191	N/A	N/A	N/A	10	N/A

***** 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, or 15). The BIOS default mapping of the PIRQx to the slave PIC is defined in Table F-4.

Table F-4 Default PIRQx to IRQx BIOS Mapping

PCI INTx	PIC IRQx
PIRQ0	IRQ11
PIRQ1	IRQ10
PIRQ2	IRQ9
PIRQ3	IRQ9

Using the interrupt steering registers of the 82371AB PIIX4, the user can override the BIOS defaults and map any of the PCI interrupts (PIRQ0-3) to any of the following PIC IRQx (ISA) interrupts: IRQ15, 14, 12-9, or 7-3.

If PCI interrupts are remapped by the user, care must be taken to ensure that all ISA and PCI functions that require an interrupt are included.

Appendix

G

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-7696 user on disk 320-500022-010, Sample Application C Code for the VMIVME-7696, included in the distribution disk set.

Because of the wide variety of environments in which the VMIVME-7696 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 SRAM

The file in this directory can be used to test the integrity of the battery backed SRAM. The additional files in this directory (Flat.c, Flat.h, Pci.c, and Pci.h) should be linked to Sram.c for compiling.

** FILE: SRAM.C

```
/*
** FILE:   SRAM.C
**
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <conio.h>
#include <dos.h>
#include “flat.h”
#include “pci.h”
#define     DID_7696     0x7696  /* Device ID              */
#define     VID_7696     0x114A  /* Vendor ID              */
void main( void )
{
 int test_int;
 unsigned long temp_dword;
 unsigned char bus, dev_func;
 FPTR sram_base;

 /* try to locate the 7696 device on the PCI bus */
 test_int = find_pci_device(DID_7696, VID_7696, 0,
            &bus, &dev_func);
 if(test_int != SUCCESSFUL)
 {
   printf(“\nUnable to locate 7696\n”);
```

```
    exit( 1 );
  }

  /* get SRAM base address from config area */
  test_int = read_configuration_area(READ_CONFIG_DWORD,
                                     bus, dev_func, 0x1C, &temp_dword);
  if(test_int != SUCCESSFUL)
  {
    printf(“\nUnable to read SRAM BASE ADDRESS @ 0x1C in config space\n”);
    exit( 1 );
  }
  sram_base = temp_dword & 0xFFFFFFF0;

  extend_seg();
  a20( 1 );

  /* Use the flat write and flat read functions to access the SRAM memory */
  fw_byte( sram_base, 0 );
  test_int = fr_byte( sram_base );

  a20(0);
  exit( 0 );

} /* end main */
```

Directory Timers

This directory contains sample code useful in the creation of applications involving the VMIVME-7696'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.

** FILE: TIMER.C

```
/*
**  FILE:   TIMER.C
*/

#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  “timers.h”

#define        DID_7696       0x7696   /* Device ID                    */
#define        VID_7696       0x114A   /* Vendor ID                    */

/* TIMERS.C function prototypes */
void far interrupt irq_rcvd( void );
void init_timer_int( void );
void restore_orig_int( void );
void load_counter( int, int, unsigned int );
void read_counter( int, int, unsigned int *, unsigned char * );
```

```
unsigned char tmr_status;
unsigned char int_line;
FPTR timers_base;

void main( void )
{
  int i;
  int test_int;
  unsigned long temp_dword;
  unsigned char bus, dev_func;

  /* try to locate the 7696 device on the PCI bus */
  test_int = find_pci_device(DID_7696, VID_7696, 0,
             &bus, &dev_func);
  if(test_int != SUCCESSFUL)
  {
    printf(“\nUnable to locate 7696\n”);
    exit( 1 );
  }
  /* get TIMERS base address from config area */
  test_int = read_configuration_area(READ_CONFIG_DWORD,
                         bus, dev_func, 0x20, &temp_dword);
  if(test_int != SUCCESSFUL)
  {
    printf(“\nUnable to read TIMERS BASE ADDRESS @ 0x20 in config space\n”);
    exit( 1 );
  }
  timers_base = temp_dword & 0xFFFFFFF0;

  /* NOTE: */
  /* the interrupt has been hard wired to ISA interrupt 5 on the 7696. */
  /* do not change the following line. */
```

```
int_line = 0x05;

extend_seg();
a20( 1 );

/* setup timers interrupt service routine */
init_timer_int();

/* set cascaded counters to independent */
fw_byte( timers_base + EXT_TMR_CNTL,
      ( TCS_B0_16 | TCS_B1_16 | TCS_B2_16 ) );

/* use the load_counter function to setup the counters */
/* load all three banks counter 1 with max (65.54 ms) */
load_counter( 0, 1, 0xFFFF );
load_counter( 1, 1, 0xFFFF );
load_counter( 2, 1, 0xFFFF );

/* read timer interrupt status register to clear any left over status */
i = fr_byte( timers_base + EXT_TMR_TIS );

/* enable all three timers and enable interrupts */
fw_byte( timers_base + EXT_TMR_ENA,
      (TE_B0_EN | TE_B1_EN | TE_B2_EN ) );

/* wait for interrupts to occur */

/* disable all three timers and interrupts */
fw_byte( timers_base + EXT_TMR_ENA,
      ( TE_B0_INT_MSK | TE_B1_INT_MSK | TE_B2_INT_MSK ) );

/* set cascaded counters to cascade for 32 bit timers */
```

```
    fw_byte( timers_base + EXT_TMR_CNTL,
        ( TCS_B0_32 | TCS_B1_32 | TCS_B2_32 ) );

    /* load all three banks counters 1 & 2 with max and one (132 ms) */
    /* counter 0 must be set to 0 */
    load_counter( 0, 0, 0x0000 );
    load_counter( 0, 1, 0xFFFF );
    load_counter( 0, 2, 0x0001 );
    load_counter( 1, 0, 0x0000 );
    load_counter( 1, 1, 0xFFFF );
    load_counter( 1, 2, 0x0001 );
    load_counter( 2, 0, 0x0000 );
    load_counter( 2, 1, 0xFFFF );
    load_counter( 2, 2, 0x0001 );

    /* read timer interrupt status register to clear any left over status */
    i = fr_byte( timers_base + EXT_TMR_TIS );

    /* enable all three timers and interrupts */
    fw_byte( timers_base + EXT_TMR_ENA,
        (TE_B0_EN | TE_B1_EN | TE_B2_EN ) );

    /* wait for interrupts to occur */

    /* disable all three timers and ints */
    fw_byte( timers_base + EXT_TMR_ENA,
        ( TE_B0_INT_MSK | TE_B1_INT_MSK | TE_B2_INT_MSK ) );

    restore_orig_int();
    a20( 0 );

} /* end main */
```

** FILE: TIMERS.C

```
/*
**  FILE: TIMERS.C
**
*/

#include  <stdlib.h>
#include  <stdio.h>
#include  <dos.h>
#include  <ctype.h>
#include  <conio.h>

#include  “timers.h”
#include  “flat.h”

#define   IRQ5       0x0D
#define   IRQ9       0x71
#define   IRQA        0x72
#define   IRQB       0x73
#define   IRQC        0x74
#define   IRQD        0x75
#define   IRQE       0x76
#define   IRQF       0x77

/* function prototypes */
void far interrupt irq_rcvd( void );
void init_timer_int( void );
void restore_orig_int( void );
void load_counter( int, int, unsigned int );
void read_counter( int, int, unsigned int *, unsigned char * );
```

```
/* global variables */
extern unsigned char int_line;
extern unsigned char tmr_status;
extern FPTR timers_base;

unsigned char pic2_org;
unsigned char pic1_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();

  /* Read 8259 slave Programmable Interrupt controller */
  pic2_org = inp(0xa1) & 0xFF; /* slave mask bits */
  pic1_org = inp(0x21) & 0xFF; /* master mask bits */
```

```
if( int_line == 0x5 )
{
  old_vect = getvect( IRQ5 ); /* save vector for IRQ 5 */
  setvect( IRQ5, irq_rcvd );

   /* enable interrupt 5 */
  outp(0x21, (pic1_org & 0xDF) );
}

else if( int_line == 0x9 )
{
  old_vect = getvect( IRQ9 ); /* save vector for IRQ 9 */
  setvect( IRQ9, irq_rcvd );

   /* enable interrupt 9 */
  outp(0xa1, (pic2_org & 0xFD) );
}

else if( int_line == 0xA )
{
  old_vect = getvect( IRQA ); /* save vector for IRQ 10 */
  setvect( IRQA, irq_rcvd );

   /* enable interrupt 10 */
  outp(0xa1, (pic2_org & 0xFB) );
}

else if( int_line == 0xB )
{
  old_vect = getvect( IRQB ); /* save vector for IRQ 11 */
  setvect( IRQB, irq_rcvd );
  /* enable interrupt 11 */
  outp(0xa1, (pic2_org & 0xF7) );
}

else if( int_line == 0xC )
{
  old_vect = getvect( IRQC );  /* save vector for IRQ 12 */
  setvect( IRQC, irq_rcvd );

  /* enable interrupt 12 */
  outp(0xa1, (pic2_org & 0xEF) );
}

else if( int_line == 0xD )
{
  old_vect = getvect( IRQD );  /* save vector for IRQ 13 */
  setvect( IRQD, irq_rcvd );

  /* enable interrupt 13 */
  outp(0xa1, (pic2_org & 0xDF) );
}

else if( int_line == 0xE )
{
  old_vect = getvect( IRQE );  /* save vector for IRQ 14 */
  setvect( IRQC, irq_rcvd );

  /* enable interrupt 14 */
  outp(0xa1, (pic2_org & 0xBF) );
}

else if( int_line == 0xF )
```

```
  {
    old_vect = getvect( IRQF ); /* save vector for IRQ 15 */
    setvect( IRQC, irq_rcvd );

    /* enable interrupt 15 */
    outp(0xa1, (pic2_org & 0x7F) );
  }

  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(0xa1, pic2_org);
outp(0x21, pic1_org);

if( int_line == 0x5 )
{
  setvect( IRQ5, old_vect );
}

else if( int_line == 0x9 )
{
  setvect( IRQ9, old_vect );
}

else if( int_line == 0xA )
{
  setvect( IRQA, old_vect );
}

else if( int_line == 0xB )
{
  setvect( IRQB, old_vect );
}

else if( int_line == 0xC )
{
  setvect( IRQC, old_vect );
}

else if( int_line == 0xD )
{
  setvect( IRQD, old_vect );
}
```

```
  else if( int_line == 0xE )
  {
    setvect( IRQE, old_vect );
  }

  else if( int_line == 0xF )
  {
    setvect( IRQF, old_vect );
  }

  enable();

} /* restore_orig_int */

/*****************************************************************/
/*  load_counter()                                              */
/*                                                              */
/*  purpose: Loads the appropriate counter in the appropriate     */
/*          bank with the count passed in.                      */
/*                                                              */
/*                                                              */
/*****************************************************************/
/*  parameters: int bank = 0, 1, 2 for BANK 0, 1, or 2           */
/*             int counter = 0, 1, 2 for COUNTER 0, 1, or 2       */
/*             unsigned int count = count to be loaded           */
/*****************************************************************/
/*  return value: none                                          */
/*****************************************************************/
void load_counter( int bank, int counter, unsigned int count )
{
  int lsb, msb;
```

```
lsb = count & 0xff;
msb = count >> 8;

switch( bank )
{
  case 0:          /* select BANK 0 */
    switch( counter )
    {
      case 0:  /* select counter 0, LSB then MSB, mode 3 */
        fw_byte( timers_base + BANK0_CNTL, (CW_SC0 | CW_LSBMSB | CW_M3) );
        fw_byte( timers_base + BANK0_CNTR0, (unsigned char) lsb );
        fw_byte( timers_base + BANK0_CNTR0, (unsigned char) msb );
      break;

      case 1:  /* select counter 1, LSB then MSB, mode 5 */
        fw_byte( timers_base + BANK0_CNTL, (CW_SC1 | CW_LSBMSB | CW_M5) );
        fw_byte( timers_base + BANK0_CNTR1, (unsigned char) lsb );
        fw_byte( timers_base + BANK0_CNTR1, (unsigned char) msb );
      break;

      case 2:  /* select counter 2, LSB then MSB, mode 5 */
        fw_byte( timers_base + BANK0_CNTL, (CW_SC2 | CW_LSBMSB | CW_M5) );
        fw_byte( timers_base + BANK0_CNTR2, (unsigned char) lsb );
        fw_byte( timers_base + BANK0_CNTR2, (unsigned char) msb );
      break;
    }
  break;

  case 1:          /* select BANK 1 */
    switch( counter )
    {
      case 0:  /* select counter 0, LSB then MSB, mode 3 */
      fw_byte( timers_base + BANK1_CNTL, (CW_SC0 | CW_LSBMSB | CW_M3) );
      fw_byte( timers_base + BANK1_CNTR0, (unsigned char) lsb );
      fw_byte( timers_base + BANK1_CNTR0, (unsigned char) msb );
    break;

    case 1: /* select counter 1, LSB then MSB, mode 5 */
      fw_byte( timers_base + BANK1_CNTL, (CW_SC1 | CW_LSBMSB | CW_M5) );
      fw_byte( timers_base + BANK1_CNTR1, (unsigned char) lsb );
      fw_byte( timers_base + BANK1_CNTR1, (unsigned char) msb );
    break;

    case 2: /* select counter 2, LSB then MSB, mode 5 */
      fw_byte( timers_base + BANK1_CNTL, (CW_SC2 | CW_LSBMSB | CW_M5) );
      fw_byte( timers_base + BANK1_CNTR2, (unsigned char) lsb );
      fw_byte( timers_base + BANK1_CNTR2, (unsigned char) msb );
    break;
  }
break;

case 2:          /* select BANK 2 */
  switch( counter )
  {
    case 0: /* select counter 0, LSB then MSB, mode 3 */
      fw_byte( timers_base + BANK2_CNTL, (CW_SC0 | CW_LSBMSB | CW_M3) );
      fw_byte( timers_base + BANK2_CNTR0, (unsigned char) lsb );
      fw_byte( timers_base + BANK2_CNTR0, (unsigned char) msb );
    break;

    case 1: /* select counter 1, LSB then MSB, mode 5 */
      fw_byte( timers_base + BANK2_CNTL, (CW_SC1 | CW_LSBMSB | CW_M5) );
      fw_byte( timers_base + BANK2_CNTR1, (unsigned char) lsb );
      fw_byte( timers_base + BANK2_CNTR1, (unsigned char) msb );
    break;

    case 2:  /* select counter 2, LSB then MSB, mode 5 */
     fw_byte( timers_base + BANK2_CNTL, (CW_SC2 | CW_LSBMSB | CW_M5) );
     fw_byte( timers_base + BANK2_CNTR2, (unsigned char) lsb );
     fw_byte( timers_base + BANK2_CNTR2, (unsigned char) msb );
    break;
   }
  break;
 }

} /* load_counter */

/*****************************************************************/
/*  read_counter()                                               */
/*                                                               */
/*  purpose: Reads the appropriate counter in the appropriate    */
/*           bank with the remaining count and status.           */
/*                                                               */
/*                                                               */
/*****************************************************************/
/*  parameters: int bank = 0, 1, 2 for BANK 0, 1, or 2           */
/*              int counter = 0, 1, 2 for COUNTER 0, 1, or 2     */
/*              unsigned int * count = remaining count           */
/*              unsigned char * status = counter status          */
/*****************************************************************/
/*  return value: none                                           */
/*****************************************************************/
void read_counter( int bank, int counter,
                   unsigned int * count, unsigned char * status )
{
 int lsb, msb;
 switch( bank )
 {
  case 0:        /* select BANK 0 */
   switch( counter )
   {
    case 0: /* select counter 0, LSB then MSB */
     fw_byte( timers_base + BANK0_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C0 ) );
     *status = fr_byte( timers_base + BANK0_CNTR0 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK0_CNTR0 ) & 0xFF;
     msb = fr_byte( timers_base + BANK0_CNTR0 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 1: /* select counter 1, LSB then MSB */
     fw_byte( timers_base + BANK0_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C1 ) );
     *status = fr_byte( timers_base + BANK0_CNTR1 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK0_CNTR1 ) & 0xFF;
     msb = fr_byte( timers_base + BANK0_CNTR1 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 2: /* select counter 2, LSB then MSB */
     fw_byte( timers_base + BANK0_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C2 ) );
     *status = fr_byte( timers_base + BANK0_CNTR2 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK0_CNTR2 ) & 0xFF;
     msb = fr_byte( timers_base + BANK0_CNTR2 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;
   }
  break;

  case 1:        /* select BANK 1 */
   switch( counter )
   {
    case 0: /* select counter 0, LSB then MSB */
     fw_byte( timers_base + BANK1_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C0 ) );
     *status = fr_byte( timers_base + BANK1_CNTR0 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK1_CNTR0 ) & 0xFF;
     msb = fr_byte( timers_base + BANK1_CNTR0 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 1: /* select counter 1, LSB then MSB */
     fw_byte( timers_base + BANK1_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C1 ) );
     *status = fr_byte( timers_base + BANK1_CNTR1 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK1_CNTR1 ) & 0xFF;
     msb = fr_byte( timers_base + BANK1_CNTR1 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 2: /* select counter 2, LSB then MSB */
     fw_byte( timers_base + BANK1_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C2 ) );
     *status = fr_byte( timers_base + BANK1_CNTR2 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK1_CNTR2 ) & 0xFF;
     msb = fr_byte( timers_base + BANK1_CNTR2 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;
   }
  break;

  case 2:        /* select BANK 2 */
   switch( counter )
   {
    case 0: /* select counter 0, LSB then MSB */
     fw_byte( timers_base + BANK2_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C0 ) );
     *status = fr_byte( timers_base + BANK2_CNTR0 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK2_CNTR0 ) & 0xFF;
     msb = fr_byte( timers_base + BANK2_CNTR0 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 1: /* select counter 1, LSB then MSB */
     fw_byte( timers_base + BANK2_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C1 ) );
     *status = fr_byte( timers_base + BANK2_CNTR1 ) & 0xFF;
     lsb = fr_byte( timers_base + BANK2_CNTR1 ) & 0xFF;
     msb = fr_byte( timers_base + BANK2_CNTR1 ) & 0xFF;
     msb = msb << 8;
     *count = ( lsb | msb );
    break;

    case 2: /* select counter 2, LSB then MSB */
     fw_byte( timers_base + BANK2_CNTL, ( CW_RBC | CW_RB_CNT |
CW_RB_STAT | CW_RB_C2 ) );
```

```
          *status = fr_byte( timers_base + BANK2_CNTR2 ) & 0xFF;
          lsb = fr_byte( timers_base + BANK2_CNTR2 ) & 0xFF;
          msb = fr_byte( timers_base + BANK2_CNTR2 ) & 0xFF;
          msb = msb << 8;
          *count = ( lsb | msb );
        break;
      }
    break;
  }

} /* read_counter */

/****************************************************************/
/*  irq_rcvd()                                            */
/*                                                    */
/*  purpose: Interrupt service routine used to service any of the  */
/*          counters on the 7696.                            */
/*                                                    */
/*                                                    */
/****************************************************************/
/*  parameters: none                                      */
/****************************************************************/
/*  return value: none                                    */
/****************************************************************/
void interrupt irq_rcvd(void)
{

  disable();

  asm {
    .386P
    push   eax
    push   ebx
  }

  /* read status */
  tmr_status = fr_byte( timers_base + EXT_TMR_TIS ) & 0xFF;

  /* Non specific end of interrupt to PIC */
  outp(0x20, 0x20);   /* Master end of irq command */

  asm {
    .386P
    pop    ebx
    pop    eax
  }

  enable();

}
```

TIMERS.H

```
/***************************************************************************/
/*                                                  */
/* FILE: TIMERS.H                                           */
/*                                                  */
/*      Header file for the 7696 timers                            */
/*                                                  */
/***************************************************************************/

#define       BANK0_CNTR0    0x00    /* Timer bank 0 counter 0          */
#define       BANK0_CNTR1    0x04    /* Timer bank 0 counter 1          */
#define       BANK0_CNTR2    0x08    /* Timer bank 0 counter 2          */
#define       BANK0_CNTL     0x0C    /* Timer bank 0 control            */
#define       BANK1_CNTR0    0x10    /* Timer bank 1 counter 0          */
#define       BANK1_CNTR1    0x14    /* Timer bank 1 counter 1          */
#define       BANK1_CNTR2    0x18    /* Timer bank 1 counter 2          */
#define       BANK1_CNTL     0x1C    /* Timer bank 1 control            */
#define       BANK2_CNTR0    0x20    /* Timer bank 2 counter 0          */
#define       BANK2_CNTR1    0x24    /* Timer bank 2 counter 1          */
#define       BANK2_CNTR2    0x28    /* Timer bank 2 counter 2          */
#define       BANK2_CNTL     0x2C    /* Timer bank 2 control            */
#define       EXT_TMR_CNTL    0x30    /* External timer control          */
#define       EXT_TMR_ENA     0x34    /* External timer enable           */
#define       EXT_TMR_TIS     0x38    /* External timer interrupt status   */

/***************************************************************************/
/* External Timer control and status register                      */
/***************************************************************************/
#define       TCS_EXP_BOOT    0x80    /* R  EXP ROM bootable - JX out      */
#define       TCS_FL_BOOT     0x40    /* R  Potable code in Flash - JX out */
#define       TCS_FL_WR_DIS   0x20    /* R  Flash write disabled - JX out  */
#define       TCS_B2_32       0x04    /* RW 1 = 32 bit cascade 0 = 16 bit  */
```

```
#define     TCS_B1_32     0x02    /* RW 1 = 32 bit cascade 0 = 16 bit  */
#define     TCS_B0_32     0x01    /* RW 1 = 32 bit cascade 0 = 16 bit  */
#define     TCS_B0_16     0x00    /* RW 1 = 32 bit cascade 0 = 16 bit  */
#define     TCS_B1_16     0x00    /* RW 1 = 32 bit cascade 0 = 16 bit  */
#define     TCS_B2_16     0x00    /* RW 1 = 32 bit cascade 0 = 16 bit  */

/**************************************************************************/
/* External Timer enable register                                 */
/**************************************************************************/
#define     TE_B0_INT_MSK   0x08    /* RW 1 = disable 0 enable int.     */
#define     TE_B1_INT_MSK   0x10    /* RW 1 = disable 0 enable int.     */
#define     TE_B2_INT_MSK   0x20    /* RW 1 = disable 0 enable int.     */
#define     TE_B2_EN        0x04    /* RW enable bank 2 timer          */
#define     TE_B1_EN        0x02    /* RW enable bank 1 timer          */
#define     TE_B0_EN        0x01    /* RW enable bank 0 timer          */

/**************************************************************************/
/* External Timer interrupt status register                        */
/**************************************************************************/
#define     TIS_B2_INT     0x04    /* RW bank 2  timer interrupt       */
#define     TIS_B1_INT     0x02    /* RW bank 1  timer interrupt       */
#define     TIS_B0_INT     0x01    /* RW bank 0  timer interrupt       */

/**************************************************************************/
/* 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      */
```

Directory VME

This directory contains the files used to setup the universe chip with one PCI-TO-VME window and enable Universe II registers to be accessed from the VME to allow mailbox access.

** FILE: 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 interrupts */
  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: 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 miscellaneous 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 - special 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 register 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 WATCHDOG

This directory contains sample code useful in the creation of applications involving the VMIVME-7696’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: WD_NMI.C

```
#include   <stdlib.h>
#include   <stdio.h>
#include   <dos.h>
#include   <time.h>
#include   <conio.h>
#include   <ctype.h>

#include   “watchdog.h”
#include   “flat.h”
#include   “pci.h”

#define      DID_7696      0x7696   /* Device ID                  */
#define      VID_7696      0x114A   /* Vendor ID                  */

/* TWRUN.C function prototypes */
void init_int( void );
void restore_orig_int( void );
void interrupt nmi_irq_rcvd( void );

/* global variables */
unsigned long int_status;
FPTR wd_base;
void far interrupt (* old_nmi_vect)(void);

void main( int argc, char * argv[] )
{
 int test_int, to_cnt;
 unsigned long temp_dword;
 unsigned char bus, dev_func;

  /* try to locate the 7696 device on the PCI bus */
 test_int = find_pci_device(DID_7696, VID_7696, 0,
            &bus, &dev_func);
 if(test_int != SUCCESSFUL)
 {
   printf(“\nUnable to locate 7696\n”);
   exit( 1 );
 }
 /* get watchdog base address from config area */
 test_int = read_configuration_area(READ_CONFIG_DWORD,
                          bus, dev_func, 0x24, &temp_dword);
 if(test_int != SUCCESSFUL)
 {
   printf(“\nUnable to read WATCHDOG BASE ADDRESS @ 0x24 in config
space\n”);
   exit( 1 );
 }
 wd_base = temp_dword & 0xFFFFFFF0;


 extend_seg();
 a20( 1 );


 /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
 fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
 fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
 fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
 fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
 fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */


 init_int();


 to_cnt = 10000;
 int_status = 0;
```

```
fw_byte( wd_base + 0x40, 0x01);  /* enable watchdog in EPLD */

fw_byte( wd_base + WD_MSEC, 0x00 ); /* 00.00 seconds */
fw_byte( wd_base + WD_SEC, 0x05 );  /* 05.00 seconds */
fw_byte( wd_base + WD_CMD, WD_TE );

do
{
  if( int_status ) break;
  delay( 1 );
  to_cnt--;

} while( to_cnt );

if( !to_cnt )
{
  printf("Timed out waiting for interrupt\n");
}

if( int_status == 1 )
{
  printf("ISR received\n");
}
else
{
  printf("ISR never entered\n");
}
fw_byte( wd_base + 0x40, 0x00 );  /* disable watchdog in EPLD */

/* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
```

```
  fw_byte( wd_base + WD_MSEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );   /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
  fw_byte( wd_base + WD_CMD, WD_WAM );  /* set watchdog alarm mask to 1 */


  restore_orig_int();
  a20( 0 );


} /* end main */


void do_exit( int xcode )
{
  exit( xcode );


}
/*****************************************************************/
/*  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 )
{
  unsigned char nmidat;
```

```
    disable();

    old_nmi_vect = getvect( 2 ); /* save vector for IRQ 09 */
    setvect( 2, nmi_irq_rcvd );

    /* arm nmi */
    nmidat = inp( 0x61 ) & 0x0F;
    nmidat |= 0x04;
    outp( 0x61, nmidat );       /* set bit 2 to clear any previous condition */
    nmidat &= 0x0B;
    outp( 0x61, nmidat );       /* clear bit 2 to enable NMI */

    /* enable nmi */
    outp( 0x70, 0x80 );
    outp( 0x70, 0x00 );

    enable();

} /* init_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 )
{
  unsigned char nmidat;

  disable();

   /* disable nmi */
  outp( 0x70, 0x80 );

  nmidat = inp( 0x61 ) & 0x0F;
  nmidat  |= 0x04;
  outp( 0x61, nmidat );        /* set bit 2 to clear any previous condition */

  setvect( 2, old_nmi_vect );

  enable();

} /* restore_orig_int */

/****************************************************************/
/*  nmi_irq_rcvd()                                      */
/*                                              */
/*  purpose: Interrupt service routine used to service nmi        */
/*          interrupts generated.                       */
/*          (NMI handler)                               */
/*                                              */
/****************************************************************/
/*  parameters: none                                    */
/****************************************************************/
```

```
/*  return value: none                              */
/******************************************************************/
void interrupt nmi_irq_rcvd( void )
{
  unsigned char nmidat;

  disable();

  int_status = 1;

   /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
  fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
  fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

   /* rearm nmi */
  nmidat = inp( 0x61 ) & 0x0F;
  nmidat |= 0x04;
  outp( 0x61, nmidat );        /* set bit 2 to clear any previous condition */
  nmidat &= 0x0B;
  outp( 0x61, nmidat );        /* clear bit 2 to enable NMI */

  enable();

}
```

** FILE: WD_RST.C

```
#include   <stdlib.h>
#include   <stdio.h>
#include   <dos.h>
#include   <time.h>
#include   <conio.h>
#include   <ctype.h>

#include   “watchdog.h”
#include   “flat.h”
#include   “pci.h”

#define      DID_7696      0x7696  /* Device ID                    */
#define      VID_7696      0x114A  /* Vendor ID                    */

void main( int argc, char * argv[] )
{
 int test_int;
 unsigned long temp_dword;
 unsigned char bus, dev_func;
 FPTR wd_base;

 /* try to locate the 7696 device on the PCI bus */
 test_int = find_pci_device(DID_7696, VID_7696, 0,
             &bus, &dev_func);
 if(test_int != SUCCESSFUL)
 {
   printf(“\nUnable to locate 7696\n”);
   exit( 1 );
 }

 /* get watchdog base address from config area */
```

```
  test_int = read_configuration_area(READ_CONFIG_DWORD,
                         bus, dev_func, 0x24, &temp_dword);
  if(test_int != SUCCESSFUL)
  {
    printf(“\nUnable to read WATCHDOG BASE ADDRESS @ 0x24 in config
space\n”);
    exit( 1 );
  }
  wd_base = temp_dword & 0xFFFFFFF0;

  extend_seg();
  a20( 1 );

  /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
  fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
  fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

  /* Jumper E18 must be installed */

  fw_byte( wd_base + 0x40, 0x02 );  /* enable watchdog in EPLD */

  fw_byte( wd_base + WD_MSEC, 0x99 ); /* 00.99 seconds */
  fw_byte( wd_base + WD_SEC, 0x99 );  /* 99.00 seconds */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_PU ) );

  /* the watchdog will time out in 99.99 seconds */
  while( !kbhit() );

  fw_byte( wd_base + 0x40, 0x00 );  /* disable watchdog in EPLD */
  /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
  fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
  fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */


  a20( 0 );


} /* end main */
```

** FILE: WD_RUN.C

```
#include   <stdlib.h>
#include   <stdio.h>
#include   <dos.h>
#include   <time.h>
#include   <conio.h>
#include   <ctype.h>

#include   “watchdog.h”
#include   “flat.h”
#include   “pci.h”

#define      DID_7696      0x7696  /* Device ID                    */
#define      VID_7696      0x114A  /* Vendor ID                    */

void main( int argc, char * argv[] )
{
  int test_int, index;
  unsigned long temp_dword;
  unsigned char bus, dev_func;
  FPTR wd_base;



  /* try to locate the 7696 device on the PCI bus */
  test_int = find_pci_device(DID_7696, VID_7696, 0,
                &bus, &dev_func);
  if(test_int != SUCCESSFUL)
  {
    printf(“\nUnable to locate 7696\n”);
    exit( 1 );
  }
```

```
 /* get watchdog base address from config area */
 test_int = read_configuration_area(READ_CONFIG_DWORD,
                                bus, dev_func, 0x24, &temp_dword);
 if(test_int != SUCCESSFUL)
 {
    printf("\nUnable to read WATCHDOG BASE ADDRESS @ 0x24 in config
space\n");
    exit( 1 );
 }
 wd_base = temp_dword & 0xFFFFFFF0;

 extend_seg();
 a20( 1 );

 /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
 fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
 fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
 fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
 fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
 fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

 printf("\nTime out set for 100 seconds. TIME OUT SHOULD NOT OCCUR\n\n");

 fw_byte( wd_base + 0x40, 0x02 );   /* enable watchdog in EPLD */

 fw_byte( wd_base + WD_MSEC, 0x99 ); /* 00.99 seconds */
 fw_byte( wd_base + WD_SEC, 0x99 );  /* 99.00 seconds */
 fw_byte( wd_base + WD_CMD, ( WD_TE | WD_PU ) );

 for( index = 500; index > 0; index-- ) {
   delay( 250 );
   /* read one of the alarm regs to cause reload */
```

```
    test_int = fr_byte( wd_base + WD_MSEC);
  }

  fw_byte( wd_base + 0x40, 0x00 );    /* disable watchdog in EPLD */

  /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
  fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0 time */
  fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

  a20( 0 );

} /* end main */
```

** FILE: WD_SF.C

```
#include    <stdlib.h>
#include    <stdio.h>
#include    <dos.h>
#include    <time.h>
#include    <conio.h>
#include    <ctype.h>

#include    “watchdog.h”
#include    “flat.h”
#include    “pci.h”

#define       DID_7696       0x7696   /* Device ID                    */
#define       VID_7696       0x114A   /* Vendor ID                    */


 /* global variables */
unsigned char bus, dev_func;
FPTR wd_base;
FPTR sys_base;

void main( int argc, char * argv[] )
{
  int test_int;
  unsigned long temp_dword;

  sys_base = 0xD800E;

   /* try to locate the 7696 device on the PCI bus */
  test_int = find_pci_device(DID_7696, VID_7696, 0,
                &bus, &dev_func);
  if(test_int != SUCCESSFUL)
```

```
{
   printf(“\nUnable to locate 7696\n”);
   exit( 1 );
}

/* get watchdog base address from config area */
test_int = read_configuration_area(READ_CONFIG_DWORD,
                                  bus, dev_func, 0x24, &temp_dword);
if(test_int != SUCCESSFUL)
{
   printf(“\nUnable to read WATCHDOG BASE ADDRESS @ 0x24 in config space\n”);
   exit( 1 );
}
wd_base = temp_dword & 0xFFFFFFF0;

extend_seg();
a20( 1 );

/* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0 time */
fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

/* Jumper E18 must be Removed */

fw_word( sys_base, 0x0100 );

fw_byte( wd_base + 0x40, 0x02 );  /* enable watchdog in EPLD */
```

```
  fw_byte( wd_base + WD_MSEC, 0x00 ); /* 00.00 seconds */
  fw_byte( wd_base + WD_SEC, 0x10 );  /* 10.00 seconds */
  fw_byte( wd_base + WD_CMD, WD_TE );

  delay( 10000 );

  /* the sys fail LED  should be on */

  while( !kbhit() );

  fw_word( sys_base, 0x0000 );

  fw_byte( wd_base + 0x40, 0x00 );  /* disable watchdog in EPLD */

  /* set WatchDog Alarm Mask 1 - deactivated and update with 0 time */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) );
  fw_byte( wd_base + WD_MSEC, 0 ); /* load with 0 to disable */
  fw_byte( wd_base + WD_SEC, 0 );  /* load with 0 to disable */
  fw_byte( wd_base + WD_CMD, ( WD_TE | WD_WAM ) ); /* allow update with 0
time */
  fw_byte( wd_base + WD_CMD, WD_WAM ); /* set watchdog alarm mask to 1 */

  a20( 0 );

} /* end main */
```

Index