

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Find the *Abaco Systems / SBS V5A* at our website: ***Click HERE***

V5A

Programmer's Manual

SBS™
Embedded Computers

SBS Embedded Computers

V5A

Pentium PC/AT Compatible VMEbus Single Board Computer

Programmer's Manual

This manual was printed on recycled paper.

Copyright © 1996 by SBS Embedded Computers Inc.

9100-31-028-A1

Document Information

V5A Programmer's Manual
Document Number **A-935-MN-03286-A1**
Part Number **9100-31-028-A1**
Revision A1

Revision History

Applicability

This manual applies to the V5A Embedded PC, revision 00 and above, until superceded.

Revision History

Revision	By	Date	Comments
A0	dlb	96.09.20	Initial release
A1	jev	98.02.19	Add SBS Logos & Layout update

Disclaimer

The information contained within this document has been carefully checked and is believed to be entirely reliable and consistent with the product that it describes. However, no responsibility is assumed for inaccuracies. Nor does SBS Embedded Computers, Inc. assume any liability arising out of the application or use of any product or circuit described herein. SBS Embedded Computers reserves the right to make changes to any product and product documentation in an effort to improve performance, reliability, or design. Furthermore, the information contained herein is of a proprietary nature and is not to be reproduced without prior written consent of SBS Embedded Computers, Inc.

Trademarks

Intel and Pentium are trademarks of Intel Corporation.

IBM, PC/AT, and AT are trademarks of International Business Machines Corporation.

MS-DOS is a trademark of Microsoft Corporation.

Conventions

A number of conventions are used throughout this manual in order to provide clarity and accuracy. These include:

- The use of an 'H' suffix to a number indicates that the number is written in hexadecimal (base sixteen) notation.
- The use of a '-' (minus) suffix to a signal name indicates that it is an active low signal. The signal is either true when the signal is at a logic zero level or the signal initiates actions on a high-to-low transition.
- Text in `Courier` Font indicates a command entry or output from an SBS embedded PC product using its built-in character set.

Contents

Chapter 1: Introduction and Manual Organization	**1-1**
Chapter Scope	1-1
Manual Scope	1-1
Manual Organization	1-1
Introduction to the V5A Software Interface	1-2
PC/AT Device Programming	1-2
VMEbus Interface Programming	1-3
The VME BIOS	1-3
Recommended Reading	1-3
Chapter 2: PC/AT Devices Programming	**2-1**
Chapter Scope	2-1
PC/AT Compatible ROM BIOS	2-1
Serial Communications Interface	2-2
Parallel Port	2-2
Keyboard Port	2-2
Real Time Clock	2-2
Non-Volatile RAM	2-3
Reset Control Register	2-3
BIOS Control Register	2-4
User Register	2-5
Serial EEPROM/Switch Register	2-5
Serial EEPROM	2-6
Serial Switch	2-7
Multi-Mode Video Controller and BIOS	2-8
V5A Memory Map	2-9
Chapter 3: VMEbus Slave Programming	**3-1**
Chapter Scope	3-1
VME Access Paths	3-1
VMEbus Slave Interface	3-1
VME Slave & PCI Master	3-1
Universe as a VMEbus Slave	3-1
Coupled Transfers	3-3
Posted Writes	3-3
Prefetched Block Reads	3-5
VMEbus Lock Commands	3-7
VMEbus Read Modify Writes	3-7
Register Accesses	3-8
DTACK Rescinding	3-8
VME Slave Images	3-8
VMEbus Fields	3-9
PCI Bus Fields	3-10
Control Fields	3-10

Configuring the VMEbus Slave using Setup 3-11
Example Slave Program 3-11

Chapter 4: VMEbus Master Programming 4-1
Chapter Scope 4-1
VMEbus Master Interface 4-1
Master Access Paths 4-1
Master Accesses in Real Mode 4-1
The Universe as VMEbus Master 4-2
Addressing Capabilities 4-3
Data Transfer Capabilities 4-3
Cycle Terminations 4-5
PCI Bus Slave Images 4-5
PCI Bus Fields 4-6
VMEbus Fields 4-7
Control Fields 4-8
Special PCI Slave Image 4-8
VME Master Example Program 4-9

Chapter 5: VMEbus Interrupt Programming 5-1
Chapter Scope 5-1
VME Interrupts and the Combined PCI Interrupt 5-1
Interrupter5-1
VMEbus Interrupt Handling 5-2
The Interrupt Channel 5-2
PCI Interrupt Generation 5-3
VMEbus Interrupt Generation 5-5
PCI Interrupt Handling 5-6
VMEbus Interrupt Handling 5-7
Bus Error During IACK 5-8
Internal Interrupt Handling 5-8
VME and PCI Software Interrupts 5-10
Software IACK Interrupt 5-11
VME Ownership Interrupt 5-11

Chapter 6: Using the VME BIOS 6-1
Chapter Scope 6-1
Introduction to the VME BIOS 6-1
Services Provided by the VME BIOS 6-1
Accessing the VME BIOS 6-2

Appendix A: Universe Registers A-1

Appendix B: Universe Block Diagram B-1

Appendix C: VME BIOS Routines C-1

Appendix D: V5A Register Summary D-1

Appendix E: VME Address Modifier Codes E-1

CHAPTER 1 Introduction and Manual Organization

Chapter Scope

This chapter introduces the software interface to the V5A Embedded PC. Also, the organization of this manual is presented.

Manual Scope

This manual is designed to aid the programmer in writing software for the V5A Embedded PC. A large portion of this manual concerns programming the VMEbus interface of the V5A. Because the V5A is 100% IBM PC/AT compatible, and because there is so much good documentation about programming the PC/AT architecture, general programming of the PC/AT section of the V5A is beyond the scope of this manual. It is assumed that you have some experience in programming a standard PC/AT. Whenever possible, however, we have suggested sources which will cover a particular topic in detail.

Manual Organization

This manual has been logically organized into six chapters and four appendices. The following paragraphs give the purpose of each chapter:

Chapter 1: Introduction and Manual Organization
The first chapter introduces the V5A software interface.

Chapter 2: PC/AT Devices Programming
This chapter describes programming the PC/AT devices of the V5A, especially those which are not found on a common PC/AT such as onboard Ethernet.

Chapter 3: VMEbus Slave Programming
This chapter describes writing software for the V5A which utilizes the VMEbus Slave interface.

Chapter 4: VMEbus Master Programming
This chapter describes writing software for the V5A which utilizes the VMEbus Master interface.

Chapter 5: VMEbus Interrupt Programming
This chapter describes using VMEbus interrupts in programs for the V5A.

Chapter 6: Using the VME BIOS
This chapter describes the VME BIOS routines and their use in a VMEbus-accessing application.

Appendix A: Universe Registers
This appendix summarizes the Universe Registers for quick, easy reference.

Appendix A: VME BIOS Routine
This appendix summarizes the VME BIOS routines for quick, easy reference.

Appendix C: Universe Block Diagram
This appendix provides a functional block diagram of the Universe VMEbus Interface Controller.

Appendix C: VME BIOS Routines
This appendix summarizes the VME BIOS routines for quick, easy reference.

Appendix D: V5A Register Reference
This appendix summarizes those registers of the V5A which are most likely to be used by user software.

Appendix E: VME Address Modifer Codes
This appendix lists the standard address modifier codes.

Introduction to the V5A Software Interface

The V5A has been designed to have as many functional elements controlled by software as possible. As such, the V5A is basically void of jumpers. All board configuration settings are completely software controlled. Therefore, software has all the responsibility of properly configuring and using the V5A. To alleviate some of the complexity of complete software control, the V5A has been designed to facilitate relatively easy programming.

PC/AT Device Programming

At the heart of the V5A is a high-performance PC/AT-compatible computer with all the common devices and interfaces one would expect from a PC/AT compatible. PCs are so widespread, and so much documentation exists for the basic PC/AT configuration, that this manual does not concentrate on the common PC/AT devices. Rather, it describes what elements are unique to the V5A: the VMEbus interface and the advanced SVGA video. However, this manual can serve as a guide for obtaining further advice on the common AT devices of the V5A. Chapter 2 discusses the PC/AT devices of the V5A, where to obtain additional information, and how to program those devices that are more advanced than those on a standard PC/AT.

VMEbus Interface Programming

An essential component in any embedded VMEbus computer card is its VMEbus interface. Since the VMEbus was primarily designed around the Motorola MC68000 CPU architecture, interfacing it to the quite different Intel 80x86 architecture is a delicate and complex task. In designing the V5A, it was our goal to provide several easy-to-use methods for communicating with the VMEbus. The V5A can function as a VMEbus Master, a VMEbus Slave, both, or neither. Because of this, details on programming the VMEbus interface have been divided into three chapters. Chapter 3 covers the Slave side of the VMEbus interface. Chapter 4 covers the Master side of the VMEbus interface. Writing programs which utilize VMEbus interrupts is covered separately in Chapter 5. The V5A can also be a VME system controller, but that function requires no programming as it can be configured using the embedded Setup program. As such, it is not discussed in detail in this manual.

The VME BIOS

The VME Basic Input Output System, or BIOS, is a collection of routines provided in ROM which provide a consistent and relatively easy means of utilizing the V5A VMEbus interface, as well as some other V5A-specific devices. The VME BIOS provides a layer of hardware abstraction which allows your software to run with little or no modification on future SBS Embedded PC products. Chapter 6 covers use of the VME BIOS when programming the V5A.

Recommended Reading

If your application requires extensive low-level programming of the V5A, or if you just need additional information about the VMEbus and the V5A's interface implementation, here are a few documents we recommend:

- VME64 ANSI/VITA 1-1994 Specification, VMEbus International Trade Association

- Universe User Manual, Tundra Semiconductor Corporation

CHAPTER 2 PC/AT Devices Programming

Chapter Scope

This chapter describes programming the PC/AT devices of the V5A. The basic peripherals such as the serial ports, the parallel port, and the keyboard port are covered generally, as are the more advanced controllers for the video, floppy disk, and hard disk interfaces. Devices not usually found on a standard desktop PC/AT, such as onboard Ethernet and advanced SVGA video, are covered in more detail.

Note that it is beyond the scope of this manual to provide the programming details associated with the IBM PC/AT compatible hardware, MS-DOS software, and the ROM BIOS. Rather, it is suggested that one of the many good programming reference books for the PC/AT architecture be consulted. We highly recommend that you obtain a copy of *Advanced MS-DOS* which is published by Microsoft Press. It presents the details of the software environment of MS-DOS and the associated calls to the ROM BIOS. If you are planning to program in 80x86 assembler or a high-level language such as 'C' and need to access system resources directly, you will find this book absolutely essential.

PC/AT Compatible ROM BIOS

The ROM BIOS is a firmware (installed in ROM) resident set of routines which is used to interface the particular system hardware to the MS-DOS operating system, and subsequently to MS-DOS applications. The use of a ROM BIOS in the original IBM PC/AT design has allowed programs to be developed specifically for the operating system environment without concern for the underlying hardware. Programs developed using the AT BIOS can be run on a variety of PC/AT compatible computers without modification. The BIOS provides a very rigid interface between the operating system and applications, and the hardware devices of the system. MS-DOS accesses devices through BIOS calls. These routines must perform their functions and return to the operating system after completion, without disturbing other system functions.

SBS Embedded Computers chose the particular ROM BIOS because it was developed for complete compatibility with IBM's BIOS, without infringing on IBM's copyright. Since the BIOS is completely compatible in operation with the original, we can assure you that programs developed correctly will run, unmodified on the V5A.

In order to use the routines contained in the AT ROM BIOS, as well as the VME BIOS, you will need to know how to make BIOS calls through software interrupts. This is described in detail in *Advanced MS-DOS*.

Serial Communications Interface

The NS16450/16550-compatible UART serial controllers within the 37C665 fully implement both the standard PC/AT COM1 and COM2 serial ports. These devices normally appear at the standard addresses of 3F8H–3FFH for COM1 and 2F8H–2FFH for COM2. These addresses are software configurable. Standard input and output with the COM ports is accomplished with BIOS calls through interrupt 14H. These calls are fully described in Section 3 of Advanced MS-DOS.

The V5A incorporates a COM2 with additional functionality. COM2 can be configured for an RS-232, RS-422, or RS-485 driver type. The COM2 driver type can be configured in the Setup utility, as well as by using bit 7 of the Serial Switch (See Serial Switch below).

WARNING: Be sure that the type of driver selected matches the type of actual connection. The differences in driver voltages and currents between the driver types may cause damage to a connection of a different driver type. Changing the COM2 configuration while the COM2 port has a connection is not recommended.

Parallel Port

The parallel port of the PC/AT is typically used for interfacing to Centronics-compatible parallel printers. MS-DOS defines the device, PRN, as the normal printer output device of the system. Therefore, the programmer should not require access to the parallel port directly. Rather, printer services available through the AT BIOS at interrupt 17H should be used. AT BIOS services are fully described in Section 3 of Advanced MS-DOS. Also, be aware that the parallel port of the V5A is capable of bi-directional communications.

Keyboard Port

The keyboard interface is designed to be fully compatible with the keyboard used on the IBM PC/AT. Built into the AT BIOS are calls for keyboard services using interrupt 16H. Section 3 of Advanced MS-DOS describes these calls in detail.

Real Time Clock

The Real-Time Clock of the V5A is equivalent to the Motorola 146818A and is contained in the DS12887 IC. The RTC occupies 3FH indexed locations and contains registers for clock timekeeping and control, as well as 16 bytes of scratchpad RAM (reserved for use by AT Setup). As long as battery power is

provided, the clock will continue updating the date and time registers as well as preserve the contents of the scratchpad RAM.

Accessing the RTC is accomplished by first writing the index address to I/O port address 70H. The valid index address range is 0-7FH. The first 40H locations are used by the V5A, while the remaining 40H are available for general use. The register or RAM data can then either be written or read at I/O port address 71H. The RTC occupies The specific use of clock registers is covered in the data sheet on the Motorola 146818A.

Notice: The RTC clock is a 100-year device. No hardware provisions exist to denote the rollover in the year 2000. An attempt to circumvent this problem have been implemented in the BIOS. However, advanced operating systems such as Windows NT do not use the BIOS to interface to the RTC. As such, they may display the year 2000 as the year 1900, etc.

Non-Volatile RAM

The Non-Volatile RAM is actually the scratchpad RAM of the RTC discussed previously. This RAM is battery-backed and contains data used by AT Setup.

Reset Control Register

The Reset Control Register is used to configure the sources and effects of reset conditions. The register is located at I/O address 2A0H.

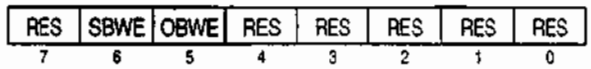


Figure 2-1 Reset Control Register (I/O 2A0H)

Bit 0: Ignore VME SYSRESET (R/W)

This bit, when set, causes the V5A to ignore incoming SYSRESET conditions from the VMEbus. This feature is handy when the V5A is being used as the development platform in a VME system. In such a system, the ability to reset the other VMEbus boards without resetting the V5A is helpful. When this bit is clear, the V5A will reset upon assertion of the VMEbus SYSFAIL signal.

Bits 1–3: Reserved

These bits are reserved for future use by SBS. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave these bits unchanged during writes.

Bit 4: Front Panel Reset to VME SYSRESET (R/W)
This bit determines whether or not the front panel Reset switch of the V5A will cause a VMEbus SYSRESET. If this bit is set, the front panel Reset switch will assert the VMEbus SYSRESET signal as well as reset the V5A. If this bit is clear, the front panel Reset switch will not assert the VMEbus SYSRESET signal and will only reset the V5A.

Bits 5–7: Reserved
These bits are reserved for future use by SBS. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave these bits unchanged during writes.

BIOS Control Register

The BIOS Control Register controls the write enables to both the onboard FLASH BIOS and the BIOS socket. The register is located at I/O address 2A1H.

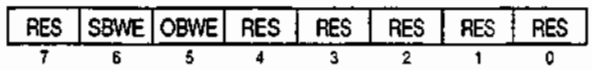


Figure 2-1 BIOS Control Register (I/O 2A1H)

Bits 0–4: Reserved
This bit is reserved for future use by SBS Embedded Computers. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave this bit unchanged during writes.

Bit 5: Onboard BIOS Write Enable (R/W)
This bit enables programming the onboard FLASH BIOS. For normal operation, this bit should be clear so that the onboard BIOS is a read-only device. It should not be set by user programs. WARNING: Setting this bit makes the BIOS suceptable to data corruption.

Bit 6: Socketed BIOS Write Enable (R/W)
This bit enables programmingof a FLASH ROM in the BIOS socket. For normal operation, this bit should be clear so that any socketed ROM is a read-only device. It should not be set by user programs. WARNING: Setting this bit makes a FLASH device in the socket suceptable to data corruption.

Bit 7: Reserved
This bit is reserved for future use by SBS Embedded Computers. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave this bit unchanged during writes.

User Register

The User Register provides a bit for controlling the SYS LED on the front panel. The purpose of this LED is user-defined.

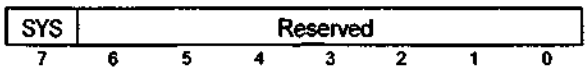


Figure 2-2 User Register (I/O 2A2H)

Bits 0–6: Reserved
These bits are reserved for future use by SBS Embedded Computers. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave these bits unchanged during writes.

Bit 7: SYS LED Control (R/W)
This bit, when set, illuminates theSYS LED on the front panel of the V5A. The purpose of this inidcator is completely user-defined. Note that readbacks of this bit will return the opposite value of the value written.

Serial EEPROM/Switch Register

The Serial EEPROM/Switch Register provides a direct interface to the program, data, and clock lines of the serially-programmable EEPROM and the serially-programmable non-volatile switch (see Serial EEPROM and Serial Switch below). Programming either of these devices requires knowledge of device-specific timing requirements, and should be done cautiously. Refer to the National Semiconductor*NM93C46 User's Manual* if you need to access this part directly.

The Serial EPROM/Switch Register is an eight-bit register at I/O address 2A3H. The bits are defined as follows:



Figure 2-3 Serial EEPROM/Switch Register (I/O 2A3H)

Bit 0: Serial Clock (R/W)
This bit is directly connected to the SK Serial Data Clock line of the NM93C46 Serial EEPROM and the SERIAL CLOCK line of the HC2001 Serial Switch. It is used for clocking serial data transfers to and from the Serial EEPROM and Serial Switch.

Bit 1: Serial Data Out (R/W)
This bit is directly connected to the DI pin of the NM93C46 and the SERIAL IN pin of the HC2001. It is used to send data to the Serial EEPROM and Serial Switch.

Bit 2: Serial Switch PROG- Control (R/W)
This bit is directly connected to the PROG- pin of the HC2001. It is used to enable a program cycle to the serial switch.

Bit 3: Serial Switch RD- Control (R/W)
This bit is directly connected to the RD- pin of the HC2001. It is used to select a read cycle from the serial switch.

Bit 4: Serial Switch Data In(R)
This bit is directly connected to the SERIAL OUT pin of the HC2001. It is used to receive serial data from the Serial Switch.

Bit 5: Serial EEPROM Data In (R)
This bit is directly connected to the DO pin of the NM93C64 during reads. It is used to receive serial data from the serial EEPROM.

Bit 6: Serial EEPROM Chip Select (R/W)
This bit is directly connected to the CS pin of the NM93C46. It selects the NM93C46 chip, making it active and ready for read or write cycles.

Bit 7: Reserved
This bit is reserved for future use by SBS Embedded Computers. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave this bit unchanged during writes.

Serial EEPROM

The V5A incorporates a 128x16 bit serially-accessed Electrically Erasable Programmable Read Only Memory, or EEPROM. This part is the National Semiconductor NM93C46, and is used for storing board setup and configuration data for SETV5A and the VME BIOS. Programming the EEPROM is facilitated through the Serial EPROM/Switch Register (see Serial EPROM/Switch Register above). However, user programs should make use of the EEPROM only under special circumstances. SBS Embedded Computers can provide you with more information about programming the EEPROM, as well as the dangers of doing so.

The EEPROM is 64 words in size. The first 43 words are used for VME setup values and board setup information. The remaining 21 words are available for general use.

Note: Like most EEPROM devices, the NM93C46 has a limited lifetime for write cycles. For the NM93C46, the average device life is 10^6 write cycles. Conservative use of the EEPROM will ensure that it lasts the life of the board.

Serial Switch

One of the major design features of the V5A is its lack of jumpers. By incorporating a serially-programmable non-volatile switch, hardware configuration is completely under software control. Most of the settings in the Serial Switch can be configured in SETV5A. Program the Serial Switch directly only when necessary.

Because of the device-specific knowledge required to successfully program it, caution should be exercised when programming the Hughes HC2001 Serial Switch directly. To read or write the Serial Switch, you must carefully follow the timing requirements specified in the Hughes HC2001/2002 data sheet. Reading or writing the lines of the Serial Switch is accomplished through the Serial EPROM/Switch Register (SP/SW). See Serial EPROM/Switch Register above.

Following are the switch definitions for the Serial Switch. For switch definitions, on is synonymous with set (one) and off is synonymous with cleared (zero):

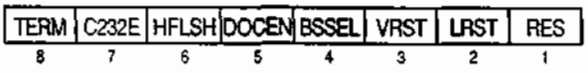


Figure 2-4 Serial Switch

Switch 1 (Bit 0): Reserved
This switch is reserved for future use by SBS Embedded Computers. Reads are not guaranteed to return any particular value. To avoid incompatibilities with future hardware, it is recommended that you leave this switch unchanged during writes.

Switch 2 (Bit 1): SYSRESET to Local Reset Enable
This switch stores the inverted startup condition of the IGSR bit of the Reset Control Register (see Reset Control Register above).

Switch 3 (Bit 2): Local Reset to VME SYSRESET Enable
This bit stores the startup condition of the FPSR bit of the Reset Control Register (see Reset Control Register above).

Switch 4 (Bit 3): BIOS Socket Select
On startup, this bit determines if the onboard FLASH BIOS or the BIOS socket will be used for the System BIOS.

Switch 5 (Bit 4): Disk-on-Chip Enable
This bit enables a portion of the BIOS region to be mapped to a Disk-On-Chip device.

Switch 6 (Bit 5): High Flash Enable
This switch swaps BIOS pages from the low region to the high region.

Switch 7 (Bit 6): COM2 RS232 Enable (COM2_RS232_EN)
This switch controls the driver type of the extended functionality COM2 port. When off, COM2 RS232 is disabled and the COM2 RS485 is enabled. When this switch is on, COM2 RS232 is enabled and the COM2 port driver type is RS232. This is the normal configuration for a PC/AT.

WARNING: Be sure that the type of driver selected matches the type of actual connection. The differences in driver voltages and currents between the driver types may cause damage to a connection of a different driver type. Changing the COM2 configuration while the COM2 port has a connection is not recommended.

Switch 8 (Bit 7): On-Board SCSI Terminator
When this switch is on, the onboard active Terminators are enabled.

Multi-Mode Video Controller and BIOS

The multi-mode video controller is implemented with a complex Large Scale Integration (LSI) device, the Cirrus Logic CL-GD543X. Primarily, you should not need to program the video controller directly, because the Video BIOS provides functions for video operation through interrupt 10H. These are fully defined in Section 3 of Advanced MS-DOS. However, when it is necessary to program the controller directly, refer to the Cirrus Logic CL-GD543X Data Sheet. Refer to table 2-1 below for the video modes supported by the V5A.

V5A Memory Map

The total accessible address space of the V5A can be divided into two parts: Figure 2-4 shows the V5A memory map and Figure 2-5 shows the V5A I/O map.

Physical Address	
FFFFFFFF 0B200000	Unused/Aliased
0B1FFFFF 0A200000	VME Access Region 4* (16M)
0A1FFFFF 09200000	VME Access Region 3* (16M)
091FFFFF 08200000	VME Access Region 2* (16M)
081FFFFF 08000000	Linear Video RAM (2M)
07FFFFFF 00100000	Local DRAM (up to 128M)
000FFFFF 000F0000	System ROM (64K)
000EFFFF 000E0000	VME Access Region 1* (64K)
000DFFFF 000D0000	Universe Registers (64K)
000CFFFF 000C8000	SCSI BIOS (32K)
000C7FFF 000C0000	Video ROM (32K)
000BFFFF 000A0000	Video RAM (128K)
0009FFFF 00000000	Local DRAM (640K)

Figure 2-4 V5A Memory Map

* These master VME access regions are shown in recommended default locations. They can be moved anywhere in the V5A memory map by changing the configuration of the access path.

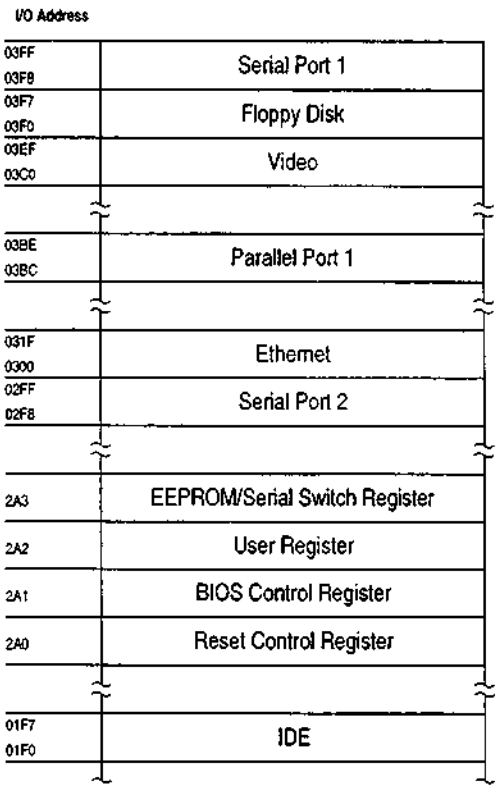


Figure 2-5 V5A I/O Map

Video Mode	Mode Type	Display Adapter	Pixel Resolution	Font Size	Chars	Colors	Dot Clock (MHz)	Horiz. Freq. (MHz)	Vert. Freq. (Hz)	Video Mem. (KB)	CRT
00h	Text	CGA	320´200	8´8	40´25	16	25	31.5	70	256	A,B,C
		EGA[2]	320´350	8´14	40´25	16	25	31.5	70	256	A,B,C
		VGA[1]	360´400	9´16	40´25	16	28	31.5	70	256	A,B,C
01h	Text	CGA	320´200	8´8	40´25	16	25	31.5	70	256	A,B,C
		EGA[2]	320´350	8´14	40´25	16	25	31.5	70	256	A,B,C
		VGA[1]	360´400	9´16	40´25	16	28	31.5	70	256	A,B,C
02h	Text	CGA	640´200	8´8	80´25	16	25	31.5	70	256	A,B,C
		EGA[2]	640´350	8´14	80´25	16	25	31.5	70	256	A,B,C
		VGA[1]	720´400	9´16	80´25	16	28	31.5	70	256	A,B,C
03h	Text	CGA	640´400	8´8	80´25	16	25	31.5	70	256	A,B,C
		EGA[2]	640´350	8´14	80´25	16	25	31.5	70	256	A,B,C
		VGA[1]	720´400	9´16	80´25	16	28	31.5	70	256	A,B,C
04h	Graph	All	320´200	8´8	40´25	4	25	31.5	70	256	A,B,C
05h	Graph	CGA	320´200	8´8	40´25	4	25	31.5	70	256	A,B,C
		EGA	320´200	8´8	40´25	4	25	31.5	70	256	A,B,C
		VGA	320´200	8´8	40´25	4	25	31.5	70	256	A,B,C
06h	Graph	All	640´200	8´8	80´25	2	25	31.5	70	256	A,B,C
07h	Text	MDA	720´350	9´14	80´25	Mono	28	31.5	70	256	A,B,C
		EGA	720´350	9´14	80´25	Mono	28	31.5	70	256	A,B,C
		VGA	720´400	9´16	80´25	Mono	28	31.5	70	256	A,B,C
08hÐ0Ch		Reserved									
0Dh	Graph	E/VGA	320´200	8´8	40´25	16	25	31.5	70	256	A,B,C
0Eh	Graph	E/VGA	640´200	8´8	80´25	16	25	31.5	70	256	A,B,C
0Fh	Graph	E/VGA	640´350	8´14	80´25	Mono	25	31.5	70	256	A,B,C
10h	Graph	E/VGA	640´350	8´14	80´25	16	25	31.5	70	256	A,B,C
11h	Graph	VGA	640´480	8´16	80´25	2	25	31.5	60	256	A,B,C
12h	Graph	VGA	640´480	8´16	80´30	16	25	31.5	60	256	A,B,C
13h	Graph	VGA	320´200	8´8	40´25	256	25	31.5	70	256	A,B,C

Chips Extended Video Modes

60h	Text	VGA	1056´400	8´16	132´25	16	40	30.5	67.5	256	A,B,C
61h	Text	VGA	1056´400	8´8	132´50	16	40	30.5	67.5	256	A,B,C
6A,70h	Graph	VGA	800´600	8´16	100´37	16	40	38	60.5	256	B,C
72h(NI)	Graph	VGA	1024´768	8´16	128´48	16	65	48.5	60	512	C
72h(I)	Graph	VGA	1024´768	8´16	128´48	16	45	35.5	86	512	A,B,C
78h	Graph	VGA	640´400	8´16	80´25	256	25	31.5	70	256	A,B,C
79h	Graph	VGA	640´480	8´16	80´30	256	25	31.5	60	512	A,B,C
7Ch	Graph	VGA	800´600	8´16	100´37	256	36	36	57	512	A,B,C

[1] Enhanced VGA mode. Otherwise, the VGA can emulate either the CGA or the EGA characteristics of this mode.
[2] The availability of these modes is dependent upon hardware and software configuration.
I = Interlaced, NI = Non-Interlaced

A = PS/2 analog CRT monitor or equivalent.
B = Multifrequency CRT (NEC Multisync 3D or equivalent)
C = Nanao Flexscan 9070s, NEC Multisync 5D or equivalent CRT monitor.

Color Palette is 256K (262,144) colors.

Table 2-1 V5A Video Modes

CHAPTER 3

VMEbus Slave Programming

Chapter Scope

This chapter describes programming the VMEbus slave interface of the V5A. Configuring the V5A as a slave, using the slave access paths, and slave registers of the Universe are covered.

VME Access Paths

The VME interface of the V5A provides four discrete paths for VMEbus data transfers. These "access paths" can be configured as either VMEbus slave access paths or VMEbus master access paths.

VMEbus Slave Interface

The VMEbus Slave Interface of the V5A is provided by the Tundra Universe VMEbus Interface Controller. The Universe presents access paths to and from VME in terms of PCI and VME "images". If an access path is configured as a slave path, then in terms of Universe functions, the path consists of a VMEbus Slave Image, a PCI Master Image, and a VMEbus Slave Channel. The Universe Block Diagram shown in Appendix B illustrates these function blocks.

VME Slave & PCI Master

In order for the V5A to be a VMEbus slave, the Universe VMEbus Interface Controller has to be a PCI bus master. As such, the remainder of this chapter is divided into two related sections, Universe as a VMEbus Slave, and Universe as a PCI Master.

Universe as a VMEbus Slave

The Universe VME Slave Channel accepts all of the addressing and data transfer modes documented in the VME64 specification (except A64 and those intended to support 3U applications, i.e. A40 and MD32). Incoming write transactions from the VMEbus may be treated as either coupled or posted, depending upon the programming of the VMEbus slave image (see "VME Slave Images" on page 3-8). With posted write transactions, data is written to a Posted Write Receive FIFO (RXFIFO), and the VMEbus master receives data acknowledg-

ment from the Universe. Write data is transferred to the PCI resource from the RXFIFO without the involvement of the initiating VMEbus master (see "Posted Writes" on page 3-3 for a full explanation of this operation). With a coupled cycle, the VMEbus master only receives data acknowledgment when the transaction is complete on the PCI bus. This means that the VMEbus is unavailable to other masters while the PCI bus transaction is executed.

Read transactions may be either prefetched or coupled. If enabled by the user, a prefetched read is initiated when a VMEbus master requests a block read transaction (BLT or MBLT) and this mode is enabled. When the Universe receives the block read request, it begins to fill its Read Data FIFO (RDFIFO) using burst transactions from the PCI resource. The initiating VMEbus master then acquires its block read data from the RDFIFO rather than from the PCI resources directly.

The Universe becomes a VMEbus slave when one of its four programmed slave images or register images are accessed by a VMEbus master (note that the Universe cannot reflect a cycle on the VMEbus and access itself). Depending upon the programming of the slave image, different possible transaction types result (see "VME Slave Images" on page 3-8 for a description of the types of accesses to which the Universe responds).

For reads, the transaction can be coupled or prefetched. Similarly, write transactions can be coupled or posted. The type of read or write transaction allowed by the slave image is dependent upon the programming of that particular VME slave image (see Figure 3-1 below and "VME Slave Images" on page 3-8). To ensure sequential consistency, prefetched reads, coupled reads, and coupled write operations are only processed once all previously posted write operations have completed (i.e. the RXFIFO is empty).

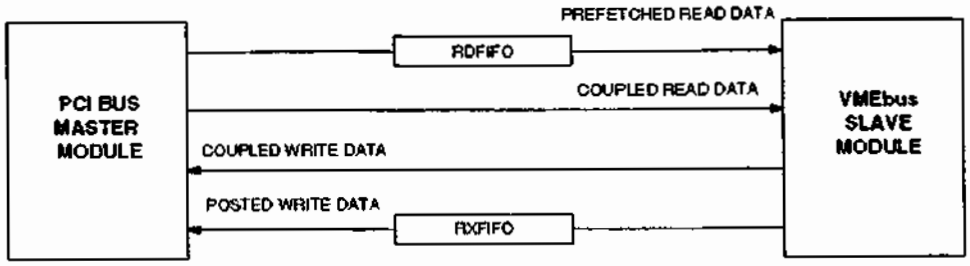


Figure 3-1 VMEbus Slave Channel Dataflow

Incoming cycles from the VMEbus can have data widths of 8-bits, 16-bits, 32-bits, and 64-bits. Although the PCI bus supports only two port sizes (32-bits and 64-bits), the byte lanes on the PCI bus can be individually enabled, which allows each type of VMEbus transaction to be directly mapped to the PCI data bus.

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Appendix A, Table A.3).

Coupled Transfers

A coupled transfer means that no FIFO is involved in the transaction and handshakes are relayed directly through the Universe. Coupled mode is the default setting for the VMEbus slave images. Coupled transfers only proceed once all posted write entries in the RXFIFO have completed (see "Posted Writes" below).

A coupled cycle with multiple data beats (i.e. block transfers) on the VMEbus side is always mapped to single data beat transactions on the PCI bus, where each data beat on the VMEbus is mapped to a single data beat transaction on the PCI bus regardless of data beat size. No packing or unpacking is performed. The only exception to this is when a D64 VMEbus transaction is mapped to D32 on the PCI bus. The data width of the PCI bus is dependent upon the programming of the VMEbus slave image (32-bit or 64-bit, see "VME Slave Images" onpage 3-8). The Universe enables the appropriate byte lanes on the PCI bus as required by the VMEbus transaction. For example, a VMEbus slave image programmed to generate 32-bit transactions on the PCI bus is accessed by a VMEbus D08 BLT read transaction (prefetching is not enabled in this slave image). The transaction is mapped to single data beat 32-bit transfers on the PCI bus with only one byte lane enabled.

The Universe does not generate RETRY* on the VMEbus, so a target-retry from a PCI slave will not be communicated to the VMEbus master. PCI transactions terminated with target abort or master abort are terminated on the VMEbus with BERR*. Note that the Universe sets the R_TA or R_MA bits in the PCI_CS register (Appendix A, Table A.3) when it receives a target abort or master abort.

Posted Writes

A posted write involves the VMEbus master writing data into the Universe's RXFIFO, rather than directly to the PCI address. Write transactions from the VMEbus are processed as posted if the PWEN bit is set in the VMEbus slave image control register (see "VME Slave Images" on page 3-8). If the bit is cleared (the default setting) the transaction bypasses the FIFO and is performed as a coupled transfer (see above). Incoming posted writes from the VMEbus are queued in a 16-entry deep RXFIFO. Each entry in the RXFIFO can contain 64 address bits (with extra bits provided in an address entry for command information), or 64 data bits. Each incoming VMEbus address phase, whether it is 16-

bit, 24-bit, or 32-bit, constitutes a single entry in the RXFIFO and is followed by subsequent data entries. The address entry contains the translated PCI address space and command information mapping relevant to the particular VMEbus slave image that has been accessed (see "VME Slave Images" on page 3-8). For this reason, any re-programming of VMEbus slave image attributes will only be reflected in RXFIFO entries queued after the re-programming. Transactions queued before the re-programming are delivered to the PCI bus with the VMEbus slave image attributes that were in use before the re-programming.

Incoming non-block write transactions from the VMEbus require two entries in the RXFIFO: one address entry (with accompanying command information) and one data entry. The size of the data entry corresponds to the data width of the VMEbus transfer. Block transfers require at least two entries: one entry for address and command information, and one or more data entries. The VME Slave Channel packs data received during block transfers to the full 64-bit width of the RXFIFO. For example, a ten data phase D16 BLT transfer (20 bytes in total) does not require ten data entries in the RXFIFO. Instead, eight of the ten data phases (16 bits per data phase for a total of 128 bits) are packed into two 64-bit data entries in the RXFIFO. The final two data phases (32 bits combined) are queued in the next RXFIFO entry. When you add the address entry to the three data entries, this VMEbus block write has been stored in a total of four RXFIFO entries.

Unlike the PCI Slave Channel (see Appendix B), the VME Slave Channel does not retry the VMEbus if the RXFIFO does not have enough space to hold an incoming VMEbus write transaction. Instead, the DTACK* response from the VME Slave Interface is delayed until space becomes available in the RXFIFO. Since single transfers require two entries in the RXFIFO, two entries must be freed up before the VME Slave Interface asserts DTACK*. Similarly, the VME Slave Channel requires two available RXFIFO entries before it can acknowledge the first data phase of a BLT or MBLT transfer (one entry for the address phase and one for the first data phase). If the RXFIFO has no available space for subsequent data phases in the block transfer, then the VME Slave Interface delays assertion of DTACK* until a single entry is available for the next data phase in the block transfer.

The availability of RXFIFO entries depends upon how many transactions are queued in the RXFIFO. The VME Slave Channel permits only four transactions queued in the RXFIFO at any one time. A transaction is a VMEbus transaction with an address phase and one or more data phases. For example a single word write is one transaction. If four single word writes are queued in the RXFIFO, then the RXFIFO is considered full. If four transactions are already queued in the RXFIFO, assertion of DTACK* is delayed until one transaction is delivered to the PCI bus.

The PCI Master Interface uses transactions queued in the RXFIFO to generate transactions on the PCI bus. No address phase deletion is performed, so the length of a transaction on the PCI bus corresponds to the length of the queued VMEbus transaction. Non-block transfers are generated on the PCI bus as single data beat transactions. Block transfers are generated as one or more burst transactions, where the length of the burst transaction is programmed by the PABS bit in the MAST_CTL register.

The Universe always packs or unpacks data from the VMEbus transaction to the PCI bus data width programmed into the VMEbus slave image (with all PCI bus byte lanes enabled). For example, consider a VMEbus slave image programmed for posted writes and a D32 PCI bus that is accessed with a VMEbus D16 block write transaction. The VMEbus D16 write transaction is mapped to a D32 write transaction on the PCI bus with all byte lanes enabled. (However, note that a single D16 transaction from the VMEbus is mapped to the PCI bus as D32 with only two byte lanes enabled).

During block transfers, the Universe will pack data to the full negotiated width of the PCI bus. This may imply that for block transfers that begin or end on addresses not aligned to the PCI bus width different byte lanes may be enabled during each data beat.

If an error occurs during a posted write to the PCI bus, the Universe uses the L_CMDERR register (Appendix A, Table A.31) to log the command information for the transaction (CMDERR [3:0]). The L_CMDERR register also records if multiple errors have occurred (with the M_ERR bit) although the actual number is not given. The error log is qualified with the L_STAT bit. The address of the errored transaction is latched in the LAERR register (Table A.32). An interrupt is generated on the VMEbus and/or PCI bus depending upon whether the VERR and LERR interrupts are enabled.

Prefetched Block Reads

Prefetching of read data occurs for VMEbus block transfers (BLT, MBLT) in those slave images that have the prefetch enable (PREN) bit set (see "VME Slave Images" on page 3-8). Without prefetching, block read transactions from a VMEbus master are handled by the VME Slave Channel as coupled reads. This means that each data phase of the block transfer is translated to a single data beat transaction on the PCI bus. In addition, only the amount of data requested during the relevant data phase is fetched from the PCI bus. For example, D16 block read transaction with 32 data phases on the VMEbus maps to 32 PCI bus transactions, where each PCI bus transaction has only two byte lanes enabled. Note the VMEbus lies idle during the arbitration time required for each PCI bus transaction, resulting in a considerable performance degradation.

With prefetching enabled, the VME Slave Channel uses a 16 entry deep RDFIFO to provide read data to the VMEbus with minimum latency. The RDFIFO is 64 bits wide, with additional bits for control information. If a VMEbus slave image is programmed for prefetching (see "VME Slave Images" on page 3-8), then a block read access to that image causes the VME Slave Channel to generate aligned burst read transactions on the PCI bus (the size of the burst read transactions is determined by the setting of the aligned burst size, PABS in the MAST_CTL register).These PCI burst read transaction are queued in the RDFIFO and the data is then delivered to the VMEbus. Note that the first data phase provided to the VMEbus master is essentially a coupled read, but subsequent data phases in the VMEbus block read are delivered from the RDFIFO and are essentially decoupled (see "Prefetched Reads" on page 3-3 for the impact on bus error handling).

The data width of the transaction on the PCI bus (32-bit or 64-bit) depends upon the setting of the LD64EN bit in the VME slave image control register (e.g. see Table A.60) and the capabilities of the accessed PCI slave. Internally, the prefetched read data is packed to 64 bits, regardless of the width of the PCI bus or the data width of the original VMEbus block read (no address information is stored with the data). Once one entry is queued in the RDFIFO, the VME Slave Interface delivers the data to the VMEbus, unpacking the data as necessary to fit with the data width of the original VMEbus block read (e.g. D16, or D32). The VME Slave Interface continuously delivers data from the RDFIFO to the VMEbus master performing the block read transaction. Because PCI bus data transfer rates exceed those of the VMEbus, it is unlikely that the RDFIFO will ever be unable to deliver data to the VMEbus master. For this reason, block read performance on the VMEbus will be similar to that observed with block writes. However, should the RDFIFO be unable to deliver data to the VMEbus master (which may happen if there is considerable traffic on the PCI bus or the PCI bus slave has a slow response) the VME Slave Interface delays DTACK* assertion until an entry is queued and is available for the VMEbus block read.

On the PCI side, prefetching continues as long as there is room for another transaction in the RDFIFO and the initiating VMEbus block read is still active. The space required in the RDFIFO for another PCI burst read transaction is determined by the setting of the PCI aligned burst size (PABS in the MAST_CTL register, Table A.56). If PABS is set for 32 bytes, there must be four entries available in the RDFIFO; for aligned burst size set to 64 bytes, eight entries must be available. When there is insufficient room in the RDFIFO to hold another PCI burst read, the read transactions on the PCI bus are terminated and only resume if room becomes available for another aligned burst AND the original VMEbus block read is still active. When the VMEbus block transfer terminates, any remaining data in the RDFIFO is purged.

Regardless of the read request, the data width of prefetching on the PCI side is full width with all byte lanes enabled. If the request is unaligned, then the first PCI data beat will have only the relevant byte lanes enabled. Subsequent data beats will have full data width with all byte lanes enabled. If LD64EN is set in the VME Slave image, the Universe requests D64 on the PCI bus by asserting REQ64# during the address phase. If the PCI slave does not respond with ACK64#, subsequent data beats are D32.

The Universe only translates errors from the PCI bus to the VMEbus during the first data beat (the coupled portion) of the prefetched read. A target abort on the PCI bus is translated as a BERR* signal on the VMEbus. However, a target abort on subsequent data beats during a prefetched read is ignored, but does cause prefetching to stop. There is no logging of the error, and all previously fetched data is provided to the VMEbus master. Once the block read on the VMEbus extends beyond the available data in the RDFIFO, then a new prefetch is initiated on the PCI bus. The point at which the new prefetch is initiated corresponds to where the error previously occurred. If the error occurs again, it will be on the first (coupled) data beat of the transaction and will be translated to the VMEbus as a BERR* signal. As with all coupled errors, no error is logged and no interrupt is generated. It should be anticipated that two target aborts may be generated before the VMEbus block read is terminated with BERR*.

VMEbus Lock Commands

The Universe supports VMEbus lock commands as described in the VME64 specification. Under the specification, ADOH cycles are used to execute the lock command (with a special AM code). Any resource locked on the VMEbus cannot be accessed by any other resource during the bus tenure of the VMEbus master. If the Universe receives a VMEbus lock command, it asserts LOCK# to the addressed resource on the PCI bus. The Universe holds the PCI bus lock until the VMEbus lock command is terminated, i.e. when BBSY* is negated. All subsequent slave VMEbus transactions are coupled while the Universe owns PCI LOCK#. Note that the VME Slave Channel has dedicated access to the PCI Master Interface during the locked transaction.

VMEbus Read Modify Writes

A read-modify-write (RMW) cycle allows a VMEbus master to read from a VMEbus slave and then write to the same resource without relinquishing bus tenure between the two operations. Each of the Universe slave images can be programmed to map RMW transactions to PCI locked transactions. If the LLRMW enable bit is set in the appropriate VMEbus slave image control register (e.g. Table A.60), then every non-block slave read is mapped to a coupled PCI locked read. LOCK# will be held on the PCI bus until AS* is negated on the VMEbus. Every non-block slave read is assumed to be a RMW since there is no possible indication from the VMEbus master that the single cycle read is just a read or the beginning of a RMW.

If the LLRMW enable bit is not set and the Universe receives a VME RMW cycle, the read and write portions of the cycle will be treated as independent transactions on the PCI bus: i.e., a read followed by a write. The write may be coupled or decoupled depending on the state of the PWEN bit in the accessed slave image.

Note: There may be a performance loss for reads that are processed through a RMW-capable slave image. Some of this comes about due to the sampling of and arbitration for LOCK# by the Universe's PCI Master Interface. More of a performance loss can arise if LOCK# is currently owned by another PCI master.

Register Accesses

The Universe registers are initially setup to appear at local addresses D000:0000 to D000:FFFF. See the Register Appendix, Page A-1, for a full description of register mapping and register access.

DTACK Rescinding

The user can increase performance in many systems by programming the Universe to rescind DTACK*. When the Universe releases the VMEbus, it can be programmed to either just release DTACK* (in which case DTACK* floats to its negated level) or rescind DTACK* (in which case the Universe actively drives DTACK* to its negated level). DTACK* rescinding is programmed by setting the RESCIND bit in the MISC_CTL register (Table A.57) and is enabled by default.

VME Slave Images

The Universe accepts accesses from the VMEbus within specific programmed slave images. Each VMEbus slave image opens a window to the resources of the PCI bus and, through its specific attributes, allows the user to control the type of access to those resources. The tables below describe programming for the VMEbus slave images by dividing them into VMEbus, PCI bus and Control fields.

Field	Register Bits	Description
base	BS[31:12] in Table A.61	multiples of 4 or 64 Kbytes (base to bound: maximum of 4 GBytes)
bound	BD[31:12] in Table A.62	
address space	VAS in Table A.60	A16, A24, A32, User 1, User 2
mode	SUPER in Table A.60	supervisor and/or non-privileged
type	PGM in Table A.60	program and/or data

Table 3-1 VMEbus Fields for VME Slave Image

Field	Register Bits	Description
image enable	EN in Table A.60	enable bit
posted write	PWEN in Table A.60	posted write enable bit
prefetched read	PREN in Table A.60	prefetched read enable bit
enable PCI D64	LD64EN in Table A.60	enables 64-bit PCI bus transactions

Table 3-2 PCI Bus Fields for VME Slave Image

Field	Register Bits	Description
translation offset	TO[31:12] in Table A.63	offsets VMEbus slave address to a selected PCI address
address space	LAS in Table A.60	Memory, I/O, Configuration
RMW	LLRMW in Table A.60	RMW enable bit

Table 3-3 Control Fields for VME Slave Image

Note: If the programming for an image is changed after the transaction is queued in the FIFO, the transaction's attributes are not changed. Only subsequent transactions are affected by the change in attributes.

VMEbus Fields

Decoding for VMEbus accesses is based on the address, and address modifiers produced by the VMEbus master. Before responding to an external VMEbus master, the address must lie in the window defined by the base and bound addresses, and the Address Modifier must match one of those specified by the address space, mode, and type fields.

The Universe's four VME slave images (images 0 to 3) are bounded by A32 space. The first of these (VME slave image 0) has a 4 Kbyte resolution while VME slave images 1 to 3 have 64-Kbyte resolution (maximum image size of 4 GBytes). Typically, image 0 would be used as an A16 image since it provides the finest granularity of the four images. Any slave image (0–3) that is enabled must have its corresponding master image disabled.

Note that the address space of a VMEbus slave image must not overlap with the address space for the Universe's control and status registers. Since register accesses take priority, any access to the overlap region results in a register access.

PCI Bus Fields

The PCI bus fields specify how the VMEbus transaction is mapped to the appropriate PCI bus transaction. The translation offset field allows the user to translate the VMEbus address to a different address on the PCI bus. The translation of VMEbus transactions beyond 4 Gbytes results in wrap-around to the low portion of the address range.

The PAS field controls generation of the PCI transaction command. The LLRMW bit allows indivisible mapping of incoming VMEbus RMW cycles to the PCI bus via the PCI LOCK# mechanism (see "VMEbus Read Modify Writes" on page 3-5). When the LLRMW bit is set, single cycle reads will always be mapped to single data beat locked PCI transactions. Setting this bit has no effect on non-block writes: they can be coupled or decoupled. However, note that only accesses to PCI Memory Space are decoupled, accesses to I/O or Configuration Space are always coupled.

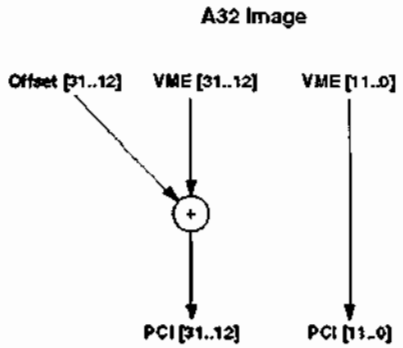


Figure 3-2 Address Translation for VMEbus to PCI Bus Transfers

Control Fields

The control fields allow the user to enable a VMEbus slave image (using the EN bit), as well as specify how reads and writes will be processed. At power-up, all images are disabled and are configured for coupled reads and writes.

If the PREN bit is set, the Universe will prefetch for incoming VMEbus block read cycles. It is the user's responsibility to ensure that prefetched reads are not destructive and that the entire image contains prefetchable resources.

If the PWEN bit is set, incoming write data from the VMEbus is loaded into the RXFIFO (see "Posted Writes" on page 3-3). Note that posted write transactions can only be mapped to Memory space on the PCI bus. Setting the PAS bit in the PCI fields to I/O or Configuration Space will force all incoming cycles to be coupled independent of this bit.

If the LD64EN bit is set, the Universe will attempt to generate 64-bit transactions on the PCI bus by asserting REQ64#. The REQ64# line is asserted during the address phase in a 64-bit PCI system, and is the means of determining whether the PCI slave is a 64-bit port. If the slave asserts ACK64# with DEVSEL#, then the Universe uses the 64-bit data bus. If the slave does not assert ACK64# with DEVSEL#, then the Universe uses a 32-bit data bus. However, note that use of REQ64# requires extra clocks internally. Therefore, if no 64-bit targets are expected on the PCI bus then performance can be improved by disabling LD64EN on the VME slave images.

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Table A.3).

Configuring the VMEbus Slave using Setup

The VMEbus Slave Interface of the V5A can be configured using the embedded Setup utility. Figure 3-2 shows the VME Access Path 0 screen in Setup in one possible configuration. In this configuration, the slave is set to appear in VME A24 space. The Slave base address is set to 400000h and the bound address is set to 500000h. The PC address will be 400000h also, as the Translation Offset is set to 000000h. In this configuration, the V5A appears as an A24 slave at VME address 400000h which will decode to local address 400000h. Refer to the V5A User's Manual for more information on accessing and using the Setup utility.

Example Slave Program

The example C program shown in Figure 3-3 demonstrates how to configure and enable the slave interface of the V5A.

```
#include <stdio.h>
#include <stdlib.h>
#include <dos.h>

#define      VERSION                    "0.90"

int image;                  // Image number
int os;                           // Image array offset

//
//  Configure the selected path as VME slave.
//

//  Initialize/Disable PCI Slave Image so Universe can become VME Slave.
//
void u_initl( unsigned int lsi_base, unsigned int lsi_bd )
{
        unsigned char far *uregs;
        unsigned char bval=0;
        uregs = MK_FP( 0xD000, 0 );
        if( image == 0 ) {
        bval = uregs[0x105+os] & 0x0f;
        uregs[0x105+os] = bval | ((lsi_base & 0xf) << 4);// Set PCI mem base
        }
        uregs[0x106+os] = (lsi_base >> 4) & 0xff;
        uregs[0x107+os] = (lsi_base >> 12) & 0xff;
        if( image == 0 ) {
        bval = uregs[0x109+os] & 0x0f;
        uregs[0x109+os] = bval | ((lsi_bd & 0xf) << 4);// Set PCI mem bound
        }
        uregs[0x10a+os] = (lsi_bd >> 4) & 0xff;
        uregs[0x10b+os] = (lsi_bd >> 12) & 0xff;
        bval = uregs[0x102+os] & 0x3f;
        uregs[0x102+os] = bval | 0x80;            // vme max width to 32 bits
        uregs[0x100+os] &= 0xfc;                  // set for pci memory space
        bval = uregs[0x403] & 0x0f;
        uregs[0x403] = bval | 0x10;               // 64 retries
        uregs[0x406] &= ~(0x19);            // syscon, etc.
        uregs[0x103+os] &= ~(0x80);               // disable pci slave image
}


//  Enable and initialize VME Slave Image.
//
void u_initv( unsigned long vsi_base, unsigned long vsi_bd )
{
        unsigned char far *uregs;
        unsigned char bval=0;      uregs = MK_FP( 0xD000, 0 );
        if( image == 0 ) {
        bval = uregs[0xf05+os] & 0x0f;
        uregs[0xf05+os] = bval | ((vsi_base & 0xf) << 4);// Set VME mem base
        }
        uregs[0xf06+os] = (vsi_base >> 4) & 0xff;
        uregs[0xf07+os] = (vsi_base >> 12) & 0xff;
        if( image == 0 ) {
        bval = uregs[0xf09+os] & 0x0f;
```

Figure 3-3 Slave Access Example Program

```
        uregs[0xf09+os] = bval | ((vsi_bd & 0xf) << 4);// Set VME mem bound
        }
        uregs[0xf0a+os] = (vsi_bd >> 4) & 0xff;
        uregs[0xf0b+os] = (vsi_bd >> 12) & 0xff;
        uregs[0xf0d+os] &= 0x0f;
        uregs[0xf0e+os] = 0;                    // Translation Offset
        uregs[0xf0f+os] = 0;
        uregs[0xf02+os] |= 0x01;          // set for A24
        uregs[0xf03+os] |= 0xe0;          // Enable VME slave image 0
                                          // +posted write enable+read
                                          // prefetch
}


void main(void)
{
      int getch(void);
      void u_initl( unsigned int, unsigned int );
      void u_initv( unsigned long, unsigned long );
      unsigned int bc, i;
      unsigned char parms, ch;
      unsigned long va;
      unsigned char la[8];
      int s;

      printf("Set for VME Slave Operation - Version %4s\n", VERSION);
      printf("Select VME Slave Image (0 - 3)\n");
      ch = getch();
      switch (ch) {
            case '0':
                  image = 0;
                  break;
            case '1':
                  image = 1;
                  break;
            case '2':
                  image = 2;
                  break;
            case '3':
                  image = 3;
                  break;
            default:
                  printf("Illegal Entry\n");
                  exit(0);
      }
      os = image * 0x14;          // 14 hex bytes between images
      u_initl(0xe0, 0xf0);              // Disable + initialize PCI slave image
      u_initv(0x01, 0xfffff1);    // Enable + initialize VME slave image
      printf("VME Slave Image %1.1i Enabled\n", image);
}
```

Figure 3-3 Slave Access Example Program

CHAPTER 4

VMEbus Master Programming

Scope

This chapter describes programming the VMEbus master interface of the V5A. Configuring the V5A as a VME master, using the four master images, and configuring the master registers of the Universe are covered.

VMEbus Master Interface

The VMEbus Master Interface of the V5A is provided by the Tundra Universe VMEbus Interface Controller. As discussed in Chapter 3, the Universe presents access paths to and from VME in terms of PCI and VME "images". If an access path is configured as a master path, then in terms of Universe functions, the path consists of a VMEbus Master Image, a PCI Slave Image, and a VMEbus Master Channel. The Universe Block Diagram shown in Appendix B illustrates these function blocks. Whenever a VME master image is enabled, the corresponding slave image must be disabled.

Master Access Paths

In Chapter 3 we discussed the four VME access paths provided by the V5A. Each of these paths can be setup to provide a local address range through which VME Master access can be made.

Master Accesses in Real Mode

Any of the four access paths can be setup to provide a "Real Mode Window", a range of address space on the V5A that provides a window to VMEbus address space for programs running in 80x86 Real Mode. It provides an easy, convenient method of accessing the VMEbus from the V5A running in 80x86 Real Mode. To use an access path to provide a Real Mode Window, configure the path with a base address of E0000 and a bound address of EFFFF. This local address range from E000:0000 to E000:FFFF will provides a 64 kilobyte page into VMEbus space. As such, it can provide a direct one-to-one map to VMEbus A16 (Short I/O) space. The Real Mode Window can be paged through VMEbus space for A24 and A32 accesses by changing the translation offset of the path.

Note: If the Real Mode Window is used, the E000 region must be excluded from any extended memory manager such as EMM386.EXE, and from Microsoft Windows. Also, be aware that calls to the Read VMEbus and Write VMEbus BIOS functions (functions 20 & 21) use the Real Mode Window E0000 region for VME access.

The Universe as VMEbus Master

The Universe becomes VMEbus master when the VME Master Interface is internally requested by the PCI Bus Slave Channel, the DMA Channel, or the Interrupt Channel. The Interrupt Channel always has priority over the other two channels. Several mechanisms are available to configure the relative priority that the PCI Bus Slave Channel and DMA Channel have over ownership of the VMEbus Master Interface.

The Universe's VME Master Interface generates all of the addressing and data transfer modes documented in the VME64 specification (except A64 and those intended to support 3U applications, i.e. A40 and MD32). The Universe is also compatible with all VMEbus modules conforming to pre-VME64 specifications. As VMEbus master, the Universe supports Read-Modify-Write (RMW), and Address-Only-with-Handshake (ADOH) but does not accept RETRY* as a termination from the VMEbus slave. The ADOH cycle is used to implement the VMEbus Lock command allowing a PCI master to lock VMEbus resources.

The Universe becomes VMEbus master as a result of the following chain of events:

1. a PCI master accesses a Universe PCI slave image (leading to VMEbus access) or the DMA Channel initiates a transaction,

2. either the Universe PCI Slave Channel or the DMA Channel wins access to the VME Master Interface through internal arbitration, and

3. the Universe Master Interface requests and obtains ownership of the VMEbus.

The Universe will also become VMEbus master if the VMEbus ownership bit is set and in its role in VMEbus interrupt handling (see Chapter 5).

The following sections describe the function of the Universe as a VMEbus master in terms of the different phases of a VMEbus transaction: addressing, data transfer, cycle termination, and bus release.

Addressing Capabilities

Depending upon the programming of the PCI slave image (see "PCI Bus Slave Images" on page 4-5), the Universe generates A16, A24, A32, and CR/CSR address phases on the VMEbus. The address mode and type (supervisor/non-privileged and program/data) are also programmed through the PCI slave image. Address pipelining is provided except during MBLT cycles, where the VMEbus specification does not permit it.

The address and AM codes that are generated by the Universe are functions of the PCI address and PCI slave image programming (see "PCI Bus Slave Images" on page 4-5) or through DMA programming.

The Universe generates ADdress-Only-with-Handshake (ADOH) cycles in support of lock commands for A16, A24, and A32 spaces. ADOH cycles must be generated through the Special Cycle Generator.

To increase the flexibility of the Universe's address space programming, there are two User Defined AM codes that can be programmed through the USER_AM register (Appendix A, Table A.59). After power-up, the two values in the USER_AM register default to the same VME64 user-defined AM code.

Data Transfer Capabilities

The data transfer between the PCI bus and VMEbus is perhaps best explained by Figure 4-1. The Universe can be seen as a funnel where the mouth of the funnel is the data width of the PCI transaction. The end of the funnel is the maximum VMEbus data width programmed into the PCI slave image. For example, consider a 32bit PCI transaction accessing a PCI slave image with VDW set to 16 bits. A data beat with all byte lanes enabled will be broken into two 16-bit cycles on the VMEbus. If the PCI slave image is also programmed with block transfers enabled, the 32-bit PCI data beat will result in a D16 block transfer on the VMEbus. Write data is unpacked to the VMEbus and read data is packed to the PCI bus data width.

If the data width of the PCI data beat is the same as the maximum data width of the PCI slave image, then the Universe maps the data beat to an equivalent VMEbus cycle. For example, consider a 32bit PCI transaction accessing a PCI slave image with VDW set to 32 bits. A data beat with all byte lanes enabled is translated to a single 32-bit cycle on the VMEbus.

As the general rule, if the PCI bus data width is less than the VMEbus data width then there is no packing or unpacking between the two buses. The only exception to this is during 32-bit PCI multi-data beat transactions to a PCI slave image programmed with maximum VMEbus data width of 64 bits. In this case, packing/unpacking occurs to make maximum use of the full bandwidth on both buses.

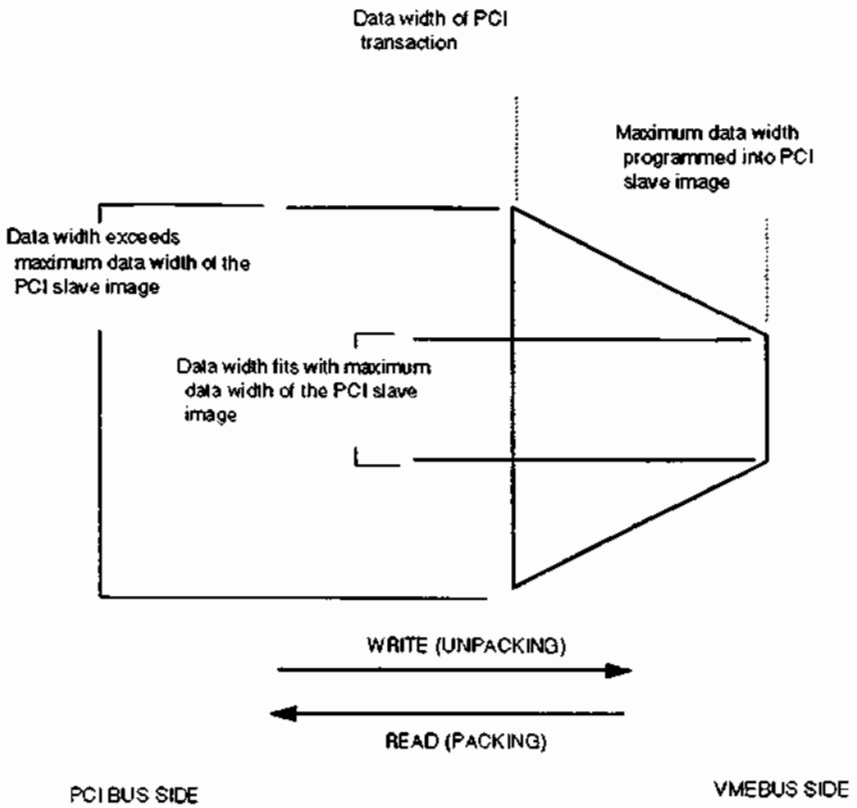


Figure 4-1 Transfer Width, Slave Image Width, & Data Packing/Unpacking

Only aligned VMEbus transactions are generated, so if the requested PCI data beat has unaligned or non-contiguous byte enables, then it is broken into multiple aligned VMEbus transactions no wider than the programmed VMEbus data width. For example, consider a three-byte PCI data beat (on a 32-bit PCI bus) accessing a PCI slave image with VDW set to 16 bits. The three-byte PCI data beat will be broken into two aligned VMEbus cycles: a single-byte cycle and a double-byte cycle (the ordering of the two cycles depends on the arrangement of the byte enables in the PCI data beat). If in the above example the PCI slave image has a VDW set to 8 bits, then the three-byte PCI data beat will be broken into three single-byte VMEbus cycles.

BLT/MBLT cycles are initiated on the VMEbus if the PCI slave image has been programmed with this capacity (see "PCI Bus Slave Images" on page 4-5). The length of the BLT/MBLT transactions on the VMEbus will be determined by the initiating PCI transaction or the setting of the PWON field in the MAST_CTL register (Appendix A, Table A.56). For example, a single data beat PCI transaction queued in the TXFIFO results in a single data beat block transfer on the VMEbus. With the PWON field, the user can specify a transfer byte count that will be dequeued from the TXFIFO before the VME Master Interface relinquishes the VMEbus.

During DMA operations, the Universe will attempt block transfers to the maximum length permitted by the VMEbus specification (256 bytes for BLT, 2 Kbytes for MBLT) and as limited by the VON counter.

Cycle Terminations

The Universe accepts BERR* or DTACK* as cycle terminations from the VMEbus slave. The assertion of BERR* indicates that some type of system error occurred and the transaction did not complete properly. A VMEbus BERR* received by the Universe during a coupled transaction is communicated to the PCI master as a target abort. No information is logged if the Universe receives BERR* in a coupled transaction. If an error occurs during a posted write to the VMEbus, the Universe uses the V_AMERR register (Table A.80) to log the AM code of the transaction (AMERR [5:0]), and the state of the IACK* signal (IACK bit, to indicate whether the error occurred during an IACK cycle). The current transaction in the FIFO is purged. The V_AMERR register also records if multiple errors have occurred (with the M_ERR bit), although the actual number of errors is not given. The error log is qualified by the value of the V_STAT bit. The address of the errored transaction is latched in the V_AERR register (Table A.81). When the Universe receives a VMEbus error during a posted write, it generates an interrupt on the VMEbus and/or PCI bus depending upon whether the VERR and LERR interrupts are enabled (see "The Interrupt Channel" on page 2-56, Table A.43 and Table A.44).

DTACK* signals the successful completion of the transaction.

PCI Bus Slave Images

The Universe accepts accesses from the PCI bus with specific programmed PCI slave images. Each image opens a window to the resources of the VMEbus and allows the user to control the type of access to those resources.The tables below describe programming for the four standard PCI bus slave images (numbered 0 to 3) by dividing them into VMEbus, PCI bus and Control fields. One special PCI slave image separate from the four discussed below is described in "Special PCI Slave Image" on page 4-8.

Field	Register Bits	Description
base	BS[31:12] in Table A.9	multiples of 4 or 64 Kbytes (base to bound: maximum of 4 GBytes)
bound	BD[31:12] in Table A.10	
address space	LAS in Table A.8	Memory, I/O, Configuration

Table 4-1 PCI Bus Fields for PCI Slave Image

Field	Register Bits	Description
translation offset	TO[31:12] in Table A.11	translates address supplied by PCI master to a specified VMEbus address
maximum data width	VDW in Table A.8	8, 16, 32, or 64 bits
address space	VAS in Table A.8	A16, A24, A32, CR/CSR, User1, User2
mode	SUPER in Table A.8	supervisor or non-privileged
type	PGM in Table A.8	program or data
cycle	VCT in Table A.8	single or block

Table 4-2 VMEbus Fields for PCI Slave Image

Field	Register Bits	Description
image enable	EN in Table A.8	enable bit
posted write	PWEN in Table A.8	enable bit

Table 4-3 Control Fields for PCI Slave Image

PCI Bus Fields

All decoding for VMEbus accesses are based on the address and command information produced by a PCI bus master. The Universe slave claims a cycle if there is an address match and if the command matches certain criteria.

All of the Universe's four PCI slave images are A32-capable only. The first of these (PCI slave image 0) has a 4 Kbyte resolution while PCI slave images 1 to 3 have 64 Kbyte resolution. Typically, image 0 would be used for an A16 image since it has the finest granularity.

Note that the address space of a PCI bus slave image must not overlap with the address space for the Universe's control and status registers. Since register accesses take priority, any access to the ovelap region results in a register access.

VMEbus Fields

The VMEbus fields map PCI transactions to a VMEbus transaction, causing the Universe to generate the appropriate VMEbus address, AM code, and cycle type. Some invalid combinations exist within the PCI slave image definition fields. For example, A16 and CR/CSR spaces do not support block transfers, and A16 space does not support 64-bit transactions. Note that the Universe does not attempt to detect or prevent these invalid programmed combinations, and that use of these combinations may cause illegal activity on the VMEbus.

The 21-bit translation offset allows the user to translate the PCI address to a different address on the VMEbus. The figure below illustrates the translation process:.

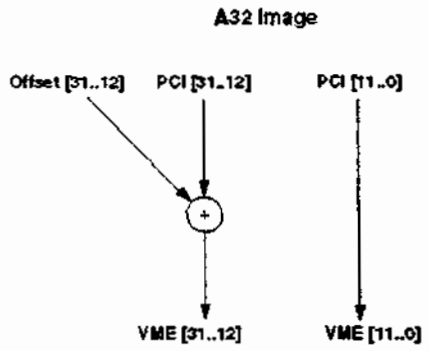


Figure 4-2 Address Translation for PCI Bus to VMEbus Transfers

The AM code generated by the Universe is a function of: the VMEbus fields, and the data width and alignment generated by the PCI bus master. For RMW and ADOH cycles, the AM code also depends upon the settings for the Special Cycle Generator.

The address space, mode, type, and cycle fields control the VMEbus AM code for most transactions. Setting the cycle field to BLT enables the BLT cycle generation. MBLT cycles are generated when the maximum width is set to 64, the BLT bit is set, and the PCI master queues a transaction with at least 64 bits (aligned) of data.

The Universe provides support for user defined AM codes. The USER_AM register (Appendix A, Table A.59) contains AM codes identified as User1 and User2. If the user selects one of these two, then the corresponding AM code from the global register is generated on the VMEbus. This approach results in standard single cycle transfers to A32 VMEbus address space independent of other settings in the VMEbus fields.

Control Fields

The control fields allow the user to enable a PCI slave image (the EN bit), as well as specify how writes are processed. If the PWEN bit is set, then the Universe will perform posted writes when that particular PCI slave image is accessed. Posted write transactions are only decoded within PCI Memory space. Accesses from other spaces will result in coupled cycles independent of the setting of the PWEN bit.

Special PCI Slave Image

The Universe provides a special PCI slave image located in Memory, I/O, or Configuration space. Its base address is aligned to 64-Mbyte boundaries and its size is fixed at 64 Mbytes (decoded using PCI address lines [31:26]). The Special PCI Slave Image is divided into four 16Mbyte regions numbered 0 to 3 (see Figure 4-2 below). These separate regions are selected with PCI address bits AD [25:24]. For example, if AD [25:24 = 01, then region 1 is decoded. Within each region, the upper 64Kbytes map to VMEbus A16 space, while the remaining portion of the 16 Mbytes maps to VMEbus A24 space. Note that no offsets are provided, so address information from the PCI transaction is mapped directly to the VMEbus.

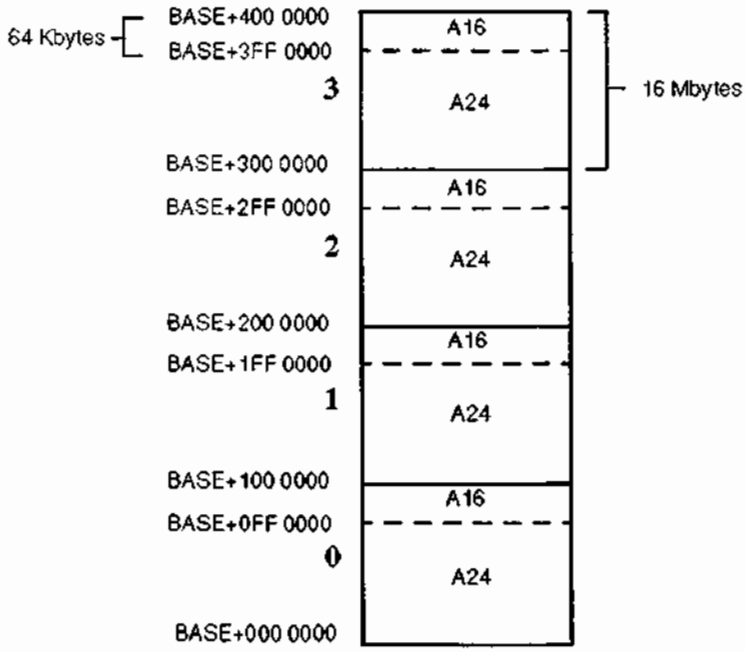


Figure 4-3 Memory Mapping in the Special PCI Slave Image

The general attributes of each region are programmed according to the tables below.

Field	Register Bits	Description
base	BS[5:0] in Table A.30	64 Mbyte aligned base address for the image
address space	LAS [1:0] in Table A.30	Places image in Memory, I/O, or Configuration space

Table 4-4 PCI Bus Fields for the Special PCI Slave Image

Field	Register Bits	Description
maximum data width	VDW in Table A.30	separately sets each region for 16 or 32 bits
mode	SUPER in Table A.30	separately sets each region as supervisor or non-privileged
type	PGM in Table A.30	separately sets each region as program or data

Table 4-5 VMEbus Fields for the Special PCI Slave Image

Field	Register Bits	Description
image enable	EN in Table A.30	enable bit for the image
posted write	PWEN in Table A.30	enable bit for posted writes for the image

Table 4-6 Control Fields for the Special PCI Slave Image

The special PCI slave image provides access to all of A16 and most of A24 space (all except the upper 64 Kbytes). By using the special PCI slave image for A16 and A24 transactions, it is possible to free the four standard PCI slave images (see "PCI Bus Slave Images" on page 3-5), which are typically programmed to access A32 space.

Note that some address space redundancy is provided in A16 space. The VMEbus specification requires only two A16 spaces, while the special PCI slave image allows for four A16 address spaces.

VME Master Example Program

The C program shown in Figure 4-4 is an example of how to configure the VMEbus master interface and perform VME master operations using the V5A. The program allows the user to select one of the four VME access paths and perform read and write operations to the VMEbus through the selected path.

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

#define VERSION                 "0.90"

int path;                       // Universe path number
int os;                               // Universe path offset
unsigned char vdw;                    // VMEbus data width

//  funct20 - Read VMEbus.
//  ENTRY:    vadd = VMEbus address
//            bytes = number of bytes to transfer
//            padd = PCI bus (local) address
//            parms =
//            2:0     VME address space: 000=A16  001=A24  010=A32
//                                       101=CR/CSR  110=User1  111=User2
//            3       pgm/data: 0=data  1=program
//            4       mode: 0=non-privileged  1=supervisor
//
//  EXIT:     Nothing.
//
void funct20( unsigned long vadd, unsigned int bytes, unsigned char *padd,
                unsigned char parms)
{
       unsigned char bval=0;
       unsigned char btmp;
       unsigned char far *uregs;          // pointer to Univ reg space
       unsigned char far *vspace;         // pointer to local mem space
       unsigned long lval;
       unsigned int i;

       uregs = MK_FP( 0xd000, 0 );
       uregs[0x102+os] = (uregs[0x102+os] & 0xf8) | (parms & 0x07);
       bval = uregs[0x101+os] & 0x0e;           // isolate PGM,SUPER clear
VCT
       btmp = (parms >> 3) & 1;
       bval |= btmp << 6;                       // position pgm bit
       btmp = (parms >> 4) & 1;
       bval |= btmp << 4;                       // position super bit
       uregs[0x101+os] = bval;                  // set them
       lval = (vadd >> 12) + 0xfff20;
       if( path == 0 ) {
       bval = uregs[0x10d] & 0xf;
       uregs[0x10d] = bval | ((lval & 0xf) << 4);
       }
       uregs[0x10e+os] = (lval >> 4) & 0xff;
       uregs[0x10f+os] = (lval >> 12) & 0xff;   // VME adr by Translation
Offset
       vspace = MK_FP( 0xe000,0 );        // VME space
       for( i=0; i<bytes; i++) {
            *padd = vspace[i + (vadd & 0xfff)];// VME to Local bus transfer
            padd++;
            }
}


//  funct21 - Write VMEbus.
//  ENTRY:    vadd = VMEbus address
```

Figure 4-4 Master Access Example

```
//            bytes = number of bytes to transfer
//            padd = PCI bus (local) address
//            parms =
//            2:0    VME address space: 000=A16  001=A24  010=A32
//                                      101=CR/CSR  110=User1  111=User2
//            3      pgm/data: 0=data  1=program
//            4      mode: 0=non-privileged  1=supervisor
//
//  EXIT:     Nothing.
//
void funct21( unsigned long vadd, unsigned int bytes, unsigned char *padd,
                unsigned char parms)
{
      unsigned char bval, btmp;
      unsigned char far *uregs;          // pointer to Univ reg space
      unsigned char far *lspace;         // pointer to local mem space
      unsigned int i;
      unsigned long lval;

      uregs = MK_FP( 0xd000, 0 );
      uregs[0x102+os] = (uregs[0x102+os] & 0xf8) | (parms & 0x07);
      bval = uregs[0x101+os] & 0x0e;           // isolate PGM,SUPER clear
VCT
      btmp = (parms >> 3) & 1;
      bval |= btmp << 6;                       // position pgm bit
      btmp = (parms >> 4) & 1;
      bval |= btmp << 4;                       // position super bit
      uregs[0x101+os] = bval;                  // set them
      lval = (vadd >> 12) + 0xfff20;
      if( path == 0) {
      bval = uregs[0x10d] & 0xf;
      uregs[0x10d] = bval | ((lval & 0xf) << 4);
      }
      uregs[0x10e+os] = (lval >> 4) & 0xff;
      uregs[0x10f+os] = (lval >> 12) & 0xff;   // VME address via T.O.
      lspace = MK_FP( 0xe000, 0 );             // VME space
      for( i=0; i<bytes; i++) {
           lspace[vadd] = *padd;             // Local bus to VMEbus xfer
           vadd++;
           padd++;
           }
}


//  Initialize the local (PCI) Slave Image so Universe can become VME Master
//  on Universe Path 0, 1, 2, or 3.
//
//  Note: The local slave image registers are 14 hex bytes apart.
//
void u_init( unsigned int lsi_base, unsigned int lsi_bd )
{
      unsigned char far *uregs;          // pointer to Univ reg space
      unsigned char bval=0;
      uregs = MK_FP( 0xD000, 0 );        // Univ reg space=mem access
      if( path == 0 ) {
      bval = uregs[0x105] & 0x0f;
      uregs[0x105] = bval | ((lsi_base & 0xf) << 4);// Set PCI mem base
      }
      uregs[0x106+os] = (lsi_base >> 4) & 0xff;
      uregs[0x107+os] = (lsi_base >> 12) & 0xff;
      if( path == 0 ) {
```

Figure 4-4 Master Access Example

```
        bval = uregs[0x109] & 0x0f;
        uregs[0x109] = bval | ((lsi_bd & 0xf) << 4);// Set PCI mem bound
        }
        uregs[0x10a+os] = (lsi_bd >> 4) & 0xff;
        uregs[0x10b+os] = (lsi_bd >> 12) & 0xff;
        bval = uregs[0x102+os] & 0x3f;
        uregs[0x102+os] = bval | vdw;          // set VMEbus max data width
        uregs[0x100+os] &= 0xfc;               // set for pci memory space
        bval = uregs[0x403] & 0x0f;
        uregs[0x403] = bval | 0x10;            // 64 retries
        bval = uregs[0x406] & 0xe2;
        uregs[0x406] = bval | 0x04;          // rescind
        uregs[0xf03+os] &= ~(0x80);          // disable vme slave image
        uregs[0x103+os] |= 0x80;               // enable pci slave image
}


void main(void)
{
      int getch(void);
      clock_t clock(void);
      void u_init( unsigned int, unsigned int);
      void funct20( unsigned long, unsigned int, unsigned char *,
      unsigned char);
      void funct21( unsigned long, unsigned int, unsigned char *,
      unsigned char);
      unsigned int bc, lc, i, j, ecnt;
      unsigned char parms, ch;
      unsigned long va;
      clock_t start, fini;
      unsigned char la[4096], db[4096];
      int ddw, s;

      printf("VME Master Access Test - Version %4s\n", VERSION);
      printf("Select Path for VME Master Transfer(0-3):\n");
      ch = getch();
      switch(ch) {
            case '0':
                  path = 0;
                  break;
            case '1':
                  path = 1;
                  break;
            case '2':
                  path = 2;
                  break;
            case '3':
                  path = 3;
                  break;
            default:
                  printf("Illegal Entry\n");
                  exit(0);
            }
      printf("Using Path %1.1i\n", path);
      printf("0 for  8 bit VMEbus data width\n");
      printf("1 for 16 bit VMEbus data width\n");
      printf("2 for 32 bit VMEbus data width\n");
      printf("3 for 64 bit VMEbus data width\n");
      printf("Select width:\n");
      ch = getch();
      switch(ch) {
      case '0':
```

Figure 4-4 Master Access Example

```
            vdw = 0 << 6;           // use 08-bit vme data width
            ddw = 8;
            break;
      case '1':
            vdw = 1 << 6;           // use 16-bit vme data width
            ddw = 16;
            break;
      case '2':
            vdw = 2 << 6;           // use 32-bit vme data width
            ddw = 32;
            break;
      case '3':
            vdw = 3 << 6;           // use 64-bit vme data width
            ddw = 64;
            break;
      default:
            printf("Illegal Entry\n");
            exit(0);
      }
      printf("Using %2.2i-bit data width\n", ddw);
      os = path * 0x14;         // 14 hex bytes between images
      lc = 1024;                // loop count
      bc = 4096;                      // transfer byte count
      for( i=0; i<bc; i++ )           // initialize data buffer
            la[i] = i;
      u_init(0xe0, 0xf0);               // Enable + initialize pci slave 0
      parms = 0x11;                     // A24 + Data + Supervisor
      while(1) {
            printf("\n0 = Quit\n");
            printf("1 = Write\n");
            printf("2 = Read\n");
            printf("3 = Compare Data\n");
            printf("Enter Option: \n");
            ch = getch();
            switch (ch) {
            case '0':           // End of program
                  printf("Bye\n");
                  exit(0);
            case '1':           // VME Master write
                  printf("Write VME address: ");
                  s = scanf("%8lx", &va);
                  if(s == 0) exit(0);
                  start = clock();
                  for( i=0; i<lc; i++ )
                        funct21( va, bc, la, parms);
                  fini = clock();
                  printf("Wt time= %f\n", (fini-start)/CLK_TCK);
                  break;
            case '2':           // VME Master read
                  printf("Read VME address: ");
                  s = scanf("%8lx", &va);
                  if(s == 0) exit(0);
                  start = clock();
                  for( i=0; i<lc; i++ )
                        funct20( va, bc, la, parms);
                  fini = clock();
                  printf("Rd Time= %f\n", (fini-start)/CLK_TCK);
                  printf(" %2.2x %2.2x %2.2x %2.2x", la[0], la[1], la[2],
la[3]);
                  printf(" %2.2x %2.2x %2.2x %2.2x\n", la[4], la[5], la[6],
la[7]);
```

Figure 4-4 Master Access Example

```
                break;
          case '3':          // VME Master WT/RD/Compare
                printf("Wt/Rd VME address: ");
                s = scanf("%8lx", &va);
                if(s == 0) exit(0);
                for(i=0; i<bc; i++)
                      db[i] = i;  // pattern compare buffer
                ecnt = 0;         // error count
                for( i=0; i<lc; i++ ) {
                      funct21( va, bc, db, parms);
                      funct20( va, bc, la, parms);
                      for(j=0; j<bc; j++) {
                            if( la[j] != db[j] ) {
                                  printf("Got %2.2x, Expected %2.2x ",
la[j], db[j]);
                                  printf("at byte %4.4x of pass %4.4x\n",
j, i);
                                  ecnt++;
                                  }
                            }
                      }
                printf("%i errors detected\n", ecnt);
                break;
          default:
                printf("Select 0,1 or 2.\n");
                break;
          }  //End switch.
    }  //End while.
}  //End main.
```

Figure 4-4 Master Access Example

CHAPTER 5 VMEbus Interrupt Programming

Chapter Scope

This chapter covers programming the interrupt facilities of the V5A VMEbus interface. Generating VMEbus interrupts, handling VMEbus interrupts, and handling local VME-related interrupts are discussed.

VME Interrups and the Combined PCI Interrupt

The Tundra Universe provides a range of VMEbus-related interrupts for use in your application. All VMEbus interrupt sources are combined to form one PCI interrupt. This PCI interrupt can be mapped to one of several local IRQ lines. IRQ5, IRQ10, IRQ11, and IRQ15 can be selected, with IRQ11 as the default. To avoid conflict with other PCI devices (SCSI, Ethernet), a PCI steerable IRQ may also be used to exclusively handle Universe interrupts. This PCI steerable interrupt can also be mapped to one of the local IRQ lines (5,10,11,15).

Interrupter

The Universe interrupt channel provides a flexible scheme to map interrupts to either the PCI bus or VMEbus interface. Interrupts are generated from either hardware or software sources (see "VMEbus Interrupt Generation" on page 5-5 for a full description of hardware and software sources). Interrupt sources can be mapped to any of the PCI bus or VMEbus interrupt output pins. Interrupt sources mapped to VMEbus interrupts are generated on the VMEbus interrupt output pins VIRQ# [7:1]. When a software and hardware source are assigned to the same VIRQn# pin, the software source always has higher priority.

Interrupt sources mapped to PCI bus interrupts are generated on one of the INT# [7:0] pins. To be fully PCI compliant, all interrupt sources must be routed to a single INT# pin.

For VMEbus interrupt outputs, the Universe interrupter supplies an 8-bit STATUS/ID to a VMEbus interrupt handler during the IACK cycle, and optionally generates an internal interrupt to signal that the interrupt vector has been provided (see "VMEbus Interrupt Generation" on page 5-5).

Interrupts mapped to PCI bus outputs are serviced by the PCI interrupt controller. The CPU determines which interrupt sources are active by reading an interrupt status register in the Universe. The source negates its interrupt when it has been serviced by the CPU (see "PCI Interrupt Generation" below).

VMEbus Interrupt Handling

A VMEbus interrupt triggers the Universe to generate a normal VMEbus IACK cycle and generate the specified interrupt output. When the IACK cycle is complete, the Universe releases the VMEbus and the interrupt vector is read by the PCI resource servicing the interrupt output. Software interrupts are ROAK, while hardware, and internal interrupts are RORA.

The Interrupt Channel

The Universe Interrupt Channel is a module within the Universe which handles prioritization and routing of various interrupt sources to interrupt outputs on the PCI bus and VMEbus. The interrupt sources are:

- the PCI LINT[7:0] lines,
- the VME IRQ*[7:1] lines,
- ACFAIL* and SYSFAIL*
- various internal signals

These sources can be routed to either the PCI LINT [7:0] lines or the VMEbus IRQ* [7:0] lines (see Figure 5-1 below). Each interrupt source is individually maskable and can be mapped to various interrupt outputs. Most interrupt sources can be mapped to one particular destination bus. The PCI sources, LINT[7:0], can only be mapped to the VMEbus interrupt outputs, while the VMEbus sources, VIRQ[7:1], can only be mapped to the PCI interrupt outputs. Some internal sources (for example, error conditions or DMA activity) can be mapped to either bus.

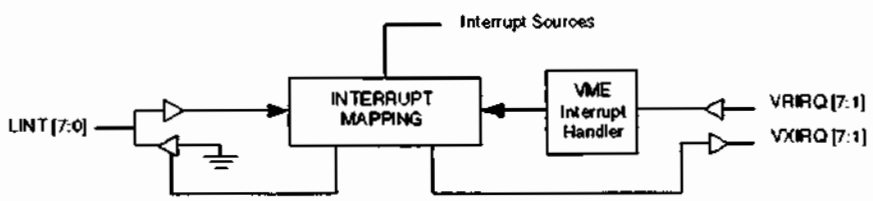


Figure 5-1 Interrupt Sources and Destinations

PCI Interrupt Generation

The Universe expands on the basic PCI specification which permits "single function" devices to assert only a single interrupt line. Eight PCI interrupt outputs are provided to give maximum flexibility, although if full PCI compliancy is required, the user may route all interrupt sources to a single PCI interrupt output.

NOTE: Only one of the PCI interrupt outputs, LINT[0], has the drive strength to be fully compliant with the PCI specification. The other seven may require buffering if they are to be routed to PCI compliant interrupt lines. For most applications, however, the drive strength provided should be sufficient.

PCI interrupts may be generated from multiple sources:

• VME interrupts
-IRQ*[7:1]
-SYSFAIL* (SYSFAIL bit)
-ACFAIL*

• internal interrupts
-DMA
-VME bus error encountered
-PCI bus error encountered
-VMEbus ownership has been granted while VOWN bit is set
-software interrupt
-VME IACK cycle performed in response to a software interrupt

Each of these sources may be individually enabled in the LINT_EN register (Appendix A, Table A.40) and mapped to a single LINT signal through the LINT_MAP0 and LINT_MAP1 registers (Appendix A, Table A.42 and Table A.43). When an interrupt is received on any of the enabled sources, the Universe asserts the appropriate LINT pin and sets a matching bit in the LINT_STAT register (Appendix A, Table A.41). See Table 5-1 below for a list of the enable, mapping and status bits for PCI interrupt sources.

Interrupt Source	Enable Bit in LINT_EN	Mapping Bit in LINT_MAP0 or LINT_MAP1	Status Bit in LINT_STAT
ACFAIL*	ACFAIL	ACFAIL	ACFAIL
SYSFAIL*	SYSFAIL	SYSFAIL	SYSFAIL
PCI Software Interrupt	SW_INT	SW_INT	SW_INT
VME Software IACK	SW_IACK	SW_IACK	SW_IACK
VMEbus Error	VERR	VERR	VERR
PCI Bus Error	LERR	LERR	LERR
DMA Event	DMA	DMA	DMA
VMEbus Interrupt Input	VIRQ7-1	VIRQ7-1	VIRQ7-1
VMEbus Ownership	VOWN	sVOWN	VOWN

Table 5-1 PCI Interrupt Source Enabling, Mapping, and Status Bits

The LINT_STAT register shows the status of all sources of PCI interrupts, independent of whether that source has been enabled. This implies that an interrupt handling routine must mask out those bits in the register that do not correspond to enabled sources on the active LINT pin.

Except for SYSFAIL* and ACFAIL*, all sources of PCI interrupts are edge sensitive. Enabling of the ACFAIL* or SYSFAIL* sources (ACFAIL and SYSFAIL bits in the LINT_EN register) cause the status bit and mapped PCI interrupt pin to assert synchronously with the assertion of the ACFAIL* or SYSFAIL* source. The PCI interrupt is negated once the ACFAIL* or SYSFAIL* source is also negated and its respective status bit in the LINT_STAT register is cleared, or the interrupt is disabled. Both of these sources are synchronized and filtered by the 64 MHz clock at their inputs.

All other sources of PCI interrupts are edge sensitive. Note that the VMEbus source for PCI interrupts actually comes out of the VMEbus Interrupt Handler block and reflects acquisition of a VME STATUS/ID. Therefore, even though VMEbus interrupts externally are level sensitive as required by the VME specification, they are internally mapped to edge-sensitive interrupts (see "VMEbus Interrupt Handling" on page 5-7).

The interrupt source status bit (in the LINT_STAT register) and the mapped LINT pin remain asserted with all interrupts. The status bit and the PCI interrupt output pin are only released when the interrupt is cleared by writing a "one" to the appropriate status bit.

VMEbus Interrupt Generation

Interrupts may be generated on any combination of VME interrupts from multiple sources:

- PCI Interrupts
  - LINT[7:0]

- Internal Interrupts
  - DMA
  - VME bus error encountered
  - PCI bus error encountered
  - software interrupt

Each of these sources may be individually enabled through the VINT_EN register (Appendix A, Table A.44) and mapped to a particular VME Interrupt level using the VINT_MAP0 and VINT_MAP1 registers (Appendix A, Table A.46 and Table A.47). Multiple sources may be mapped to any VME level. Mapping interrupt sources to level 0 effectively disables the interrupt.

Once an interrupt has been received from any of the sources, the Universe sets the corresponding status bit in the VINT_STAT register (Appendix A, Table A.45), and asserts the appropriate VMEbus interrupt output signal (if enabled). When a VME interrupt handler receives the interrupt, it will perform an IACK cycle at that level. When the Universe decodes that IACK cycle together with IACKIN* asserted, it provides the STATUS/ID previously stored in the STATID register (Appendix A, Table A.48). See Table 5-2 below for a list of the enable, mapping and status bits for VMEbus interrupt sources.

Interrupt Source	Enable Bit in VINT_EN	Mapping Bit in VINT_MAP0 or VINT_MAP1	Status Bit in VINT_STAT
VME Software Interrupt	SW_INT	SW_INT	SW_INT
VMEbus Error	VERR	VERR	VERR
PCI Bus Error	LERR	LERR	LERR
DMA Event	DMA	DMA	DMA
PCI bus Interrupt Input	LINT7-0	LINT7-0	LINT7-0

Table 5-2 VMEbus Interrupt Source Enabling, Mapping, and Status Bits

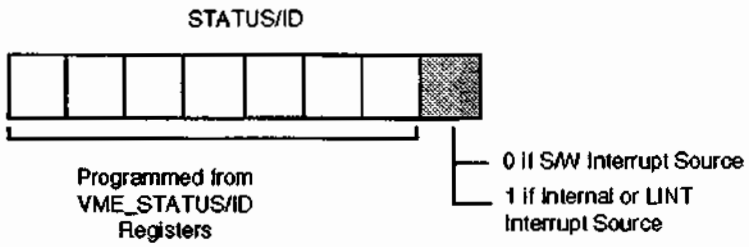


Figure 5-2 Status/ID Provided by the Universe

For all VMEbus interrupts, the Universe interrupter supplies a pre-programmed 8-bit STATUS/ID: a common value for all interrupt levels. The upper seven bits are programmed in the STATID register. The lowest bit is cleared if the source of the interrupt was the software interrupt, and is set for all other interrupt sources. If a software interrupt source and another interrupt source are active and mapped to the same VMEbus interrupt level, the Universe gives priority to the software source.

Once the Universe has provided the STATUS/ID to an interrupt handler during a software initiated VMEbus interrupt, it generates an internal interrupt, SW_IACK. If enabled, this interrupt feeds back to the PCI bus (through one of the LINT pins) to signal a process that the interrupt started through software has been completed.

All VMEbus interrupts generated by the Universe are RORA, except for the software interrupt which is ROAK. This means that if the interrupt source was the software interrupt, then the VMEbus interrupt output is automatically negated when the Universe receives the IACK cycle. However, for any other interrupt, the VMEbus interrupt output remains asserted until cleared by a register access. Writing a "one" to the relevant bit in the VINT_STAT register clears that interrupt source. However, since PCI interrupts are level sensitive, if an attempt is made to clear the VMEbus interrupt while the LINT pin is still asserted, the VMEbus interrupt remains asserted. This causes a second interrupt to be generated to the VMEbus. For this reason, a VME interrupt handler should clear the source of the PCI interrupt before clearing the VMEbus interrupt.

PCI Interrupt Handling

All eight PCI interrupt lines, LINT[7:0], can act as interrupt inputs to the Universe. They are level sensitive and, if enabled in the VINT_EN register (Table A.44), immediately generate an interrupt to the VMEbus. It is expected that when a VMEbus interrupt handler receives the Universe's STATUS/ID from the Universe, the interrupt handler will clear the VMEbus interrupt by first clearing the source of the interrupt on the PCI bus, and then clearing the VMEbus interrupt itself (by writing a "one" to the appropriate bit in the VINT_STAT register, Table A.45).

Note that since PCI interrupts are level sensitive, if an attempt is made to clear the VMEbus interrupt while the LINT pin is still asserted, the VMEbus interrupt

remains asserted. This causes a second interrupt to be generated to the VMEbus. For this reason, a VMEbus interrupt handler should clear the source of the PCI interrupt before clearing the VMEbus interrupt.

VMEbus Interrupt Handling

As a VME interrupt handler, the Universe can monitor any or all of the VMEbus interrupt levels. It can also monitor SYSFAIL* and ACFAIL*, although IACK cycles are not generated for these inputs. Each interrupt is enabled through the LINT_EN register (Appendix A, Table A.40).

Once enabled, assertion of any of the VMEbus interrupt levels, IRQ[7:1]*, causes the internal interrupt handler circuitry to request ownership of the Universe's VME Master Interface on the level programmed in the MAST_CTL register . This interface is shared between several channels in the Universe: the PCI Slave Channel, the DMA Channel, and the Interrupt Channel. The Interrupt Channel has the highest priority over all other channels and, if an interrupt is pending, assumes ownership of the VME Master Interface when the previous owner has relinquished ownership.

There may be some latency between reception of a VMEbus interrupt and generation of the IACK cycle. This arises because of the latency involved in the Interrupt Channel gaining control of the VME master interface, and because of possible latency in gaining ownership of the VMEbus if the VME Master Interface is programmed for release-when-done. In addition, the Universe only generates an interrupt on the PCI bus once the IACK cycle has completed on the VMEbus. Because of these combined latencies (time to acquire VMEbus and time to run the IACK cycle), systems should be designed to accommodate a certain worst case lag time from VMEbus interrupt generation to its translation to the PCI bus.

When the Universe receives a STATUS/ID in response to an IACK cycle, it stores that value in one of seven registers. These registers, V1_STATID through V7_STATID (Appendix A, Table A.49 to Table A.55), store the STATUS/ID corresponding to each IACK level (in the STATID field). Once an IACK cycle has been generated and the resulting STATUS/ID is latched, another IACK cycle will not be run on that level until the level has been re-armed by writing a "one" to the corresponding status bit in the VINT_STAT register (Table A.45). If other interrupts (at different levels) are pending while the interrupt is waiting to be re-armed, IACK cycles are run on those levels in order of priority and the STATUS/IDs stored in their respective registers.

Once the IACK cycle is complete and the STATUS/ID stored, an interrupt is generated to the PCI bus on one of LINT#[7:0] interrupt output lines depending on the mapping for that VME level in the LINT_MAP0 register. The interrupt is cleared and the VMEbus interrupt level is re-armed by clearing the correct bit in the LINT_STAT register.

Bus Error During IACK

A bus error encountered on the VMEbus while the Universe is performing an IACK cycle is handled by the Universe in two ways. The first is through the error logs in the VME Master Interface. These logs store address and command information whenever the Universe encounters a bus error on the VMEbus . If the error occurs during an IACK cycle, the IACK# bit is set in the V_AMERR register (Appendix A, Table A.80). The VME Master Interface also generates an internal interrupt to the Interrupt Channel indicating a VMEbus error occurred. This internal interrupt can be enabled and mapped to either the VMEbus or PCI bus.

As well as generating an interrupt indicating an error during the IACK cycle, the Universe also generates an interrupt as though the IACK cycle completed successfully. If an error occurs during the fetching of the STATUS/ID, the Universe sets the ERR bit in the Vn_STATID register (Table A.49 to Table A.55), and generates an interrupt on the appropriate LINT pin (as mapped in the LINT_MAP0 register, Table A.42). The PCI resource, upon receiving the PCI interrupt, is expected to read the STATUS/ID register, and take appropriate actions if the ERR bit is set. Note that the STATUS/ID cannot be considered valid if the ERR bit is set in the STATUS/ID register.

It is important to recognize that the IACK cycle error may generate two PCI interrupts: one through the VMEbus master bus error interrupt and another through the standard PCI interrupt translation. Should an error occur during acquisition of a STATUS/ID, the VINT_STAT register (Table A.45) will show that both VIRQx, and VERR are active.

Internal Interrupt Handling

The Universe's internal interrupts are routed from several processes in the device (see Figure 5-3 on page 5-9). There is an interrupt from the VME master interface to indicate a VMEbus error, another from the PCI master interface to indicate an error on that bus, another from the DMA to indicate various conditions in that channel, along with several others as indicated in Table 5-3 below. Some interrupts may be routed only to the PCI bus, some only to the VMEbus, and others that may be routed to both (although we recommend that you map interrupts to a single bus).

Interrupt Source	May be Routed to:	
	VMEbus	PCI Bus
PCI bus errors	√	√
VME bus error	√	√
DMA event	√	√
IACK cycle complete for s/w interrupt		√
VME bus ownership granted		√
PCI s/w interrupt		√
VME s/w interrupt	√	

Table 5-3 Internal Interrupt Routing

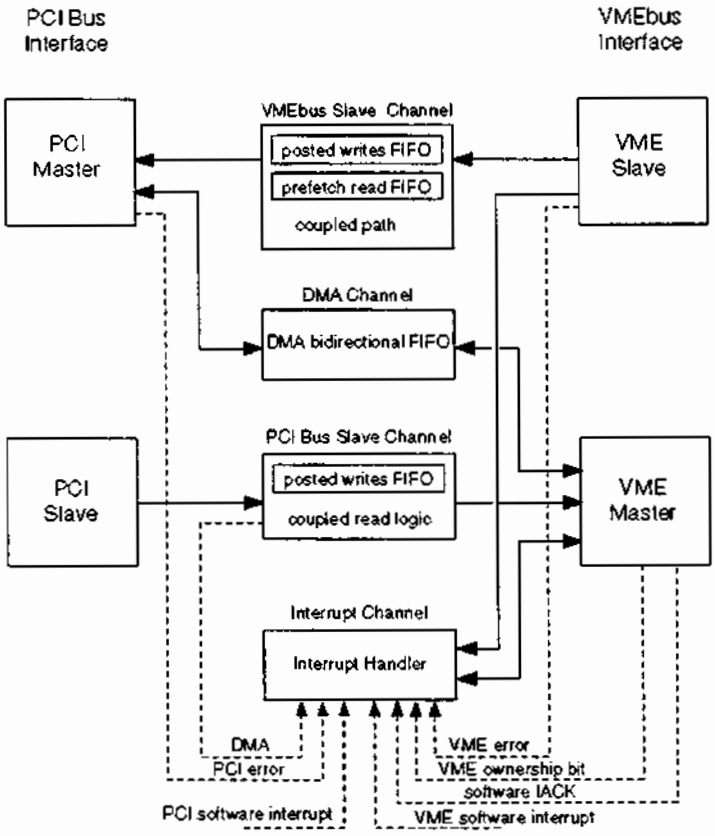


Figure 5-3 Sources of Internal Interrupts

VME and PCI Software Interrupts

Two interrupt sources are provided as mechanisms to interrupt the PCI or VME buses through software. These interrupts may be triggered by writing a "one" to the respective enable bits. Writing a "one" to the SW_INT bit in the VINT_EN register (Appendix A, Table A.44) triggers one (and only one) interrupt on the VMEbus on the level programmed in the VINT_MAP1 register (Appendix A, Table A.47). When an IACK cycle is serviced on the VMEbus, the Universe can be programmed to generate an interrupt on the PCI bus by setting the SW_IACK enable bit in the LINT_EN register. Writing a "one" to the SW_INT in the LINT_EN (Appendix A, Table A.40) registersimilarly generates an interrupt to the PCI bus. While these interrupt sources are active, their respective status bits are set (SW_INT in VINT_STAT; SW_IACK and SW_INT in LINT_STAT). Either interrupt may be cleared by clearing the respective enable bit although clearing a VMEbus interrupt through this method is not recommended since it may result in a spurious interrupt on that bus.

Since the software interrupt is edge-sensitive, the SW_INT bit should be cleared before generating another interrupt. It is recommended that the appropriate interrupt handler clear this bit once it has completed its operations. Alternatively, the process generating the interrupt could clear this bit before re-asserting it.

Software interrupts on the VMEbus have priority over other interrupts mapped internally to the same level on the VMEbus. When a VMEbus interrupt handler generates an IACK cycle on a level mapped to both a software interrupt and another interrupt, the Universe always provides the STATUS/ID for the software interrupt (bit zero of the Status/ID is cleared). If there are no other active interrupts on that level, the interrupt is automatically cleared upon completion of the IACK cycle (since software interrupts are ROAK).

While the software interrupt STATUS/ID has priority over other interrupt sources, the user can give other interrupt sources priority over the software interrupt. This is done by reading the LINT_STAT register (Table A.41) when handling a Universe interrupt. This register indicates all active interrupt sources. Using this information, the interrupt handler can then handle the interrupt sources in any system-defined order.

Software IACK Interrupt

The Universe generates an internal interrupt when it provides the software STATUS/ID to the VMEbus. This interrupt can only be routed to a PCI interrupt output. A PCI interrupt will be generated upon completion of an IACK cycle that had been initiated by the Universe's software interrupt if the SW_IACK bit in the LINT_EN register (Appendix A, Table A.40) is set and the SW_IACK field in the LINT_MAP1 register (Appendix A, Table A.43) is mapped to a corresponding PCI interrupt line. This interrupt could be used by a PCI process to indicate that the software interrupt generated to the VMEbus has been received by the slave device and acknowledged.

Like other interrupt sources, this interrupt source can be independently enabled through the LINT_EN register (Appendix A, Table A.40) and mapped to a particular LINT pin using the LINT_MAP1 register (Appendix A, Table A.43). A status bit in the LINT_STAT register (Appendix A, Table A.41) indicates when the interrupt source is active, and is used to clear the interrupt once it has been serviced.

VME Ownership Interrupt

The VMEbus ownership interrupt is generated when the Universe acquires the VMEbus in response to programming of the VOWN bit in the MAST_CTL register (Appendix A, Table A.56). This interrupt source can be used to indicate that ownership of the VMEbus is ensured during an exclusive access. The interrupt is cleared by writing a one to the matching bit in the LINT_STAT register (Appendix A, Table A.41).

CHAPTER 6 Using the VME BIOS

Chapter Scope

This chapter describes the purpose and use of the V5A onboard VME BIOS

Introduction to the VME BIOS

The VME Basic Input/Output System (BIOS) is a collection of routines in ROM which allow software to easily take advantage of the V5A VMEbus interface. In previous chapters, the specific operation and programming of the V5A VMEbus interface was described. For most applications, however, the VME BIOS provides a much more convenient method of using the V5A VMEbus interface. You should only need to program the interface directly under the following circumstances:

1. You require absolute control of board operations due to timing constraints, etc.

2. You require more flexibility in the use of the VME interface such as use of the master or slave images in 80x86 Protected Mode.

3. Your software runs in 80x86 protected mode.

Services Provided by the VME BIOS

The VME BIOS provides many of the services a programmer would commonly need to write a VMEbus-accessing application. Services provided by the VME BIOS, divided into functional groups, are shown below:

VMEbus Interrupt Services

Request VMEbus Interrupt, Get VMEbus Interrupt Status, Clear VME Interrupts, Service Interrupts

VMEbus Slave Services
Get Slave Image, Put Slave Image

VMEbus Master Services
Get Master Image, Put Master Image, ReadVMEbus, Write VMEbus

Board Services
Get VME BIOS Version, Set BoardInfo, Get SYSFAIL, Set SYSFAIL

Accessing the VME BIOS

Accessing the VME BIOS is relatively simple because calls to it are made just like calls to any other BIOS. Accessible through software interrupt 69H, the individual BIOS routine you want is specified in register AH. Detailed information about the numbers of VME BIOS routines, required parameters, and an in-depth functional description of each is provided in Appendix C of this manual.

APPENDIX A Universe Registers

Appendix Scope

This appendix provides the register descriptions, bit definitions, and locations of the registers in the Universe VMEbus Interface Controller. The information in this appendix is a direct excerpt from Tundra Semiconductor Corporation's Universe User Manual. All page references, table numbers, and figure numbers refer to the Universe User Manual, and not to this Programmer's Manual.

The Universe Control and Status Registers (UCSR) facilitate host system congur-ation and allow the user to control Universe operational characteristics. The UCSRs are divided into three groups:

- PCI Conguration Space (PCICS),
- VMEbus Control and Status Registers (VCSR), and
- Universe Device Specic Status Registers (UDSR).

The Universe registers are little-endian.

Figure A.1 below summarizes the supported register access mechanisms.

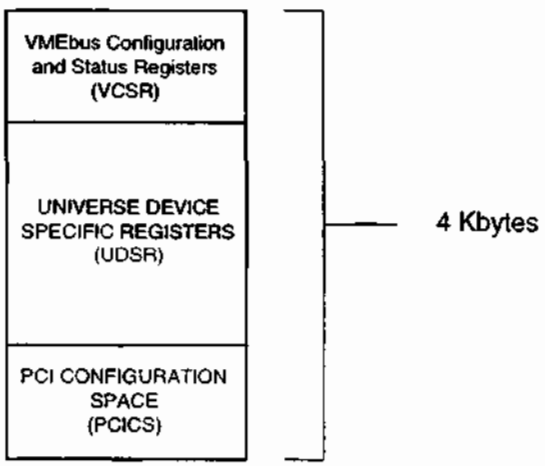


Figure A.1 : UCSR Access Mechanisms

Table A.1 below lists the Universe registers by address offset. The tables following the register map (Table A.2 to Table A.84) provide detailed descriptions of each register.

Address offsets in Table A.1 below apply to accesses from the PCI bus and to accesses from the VMEbus side using the VMEbus Register Access Image (see "Registers" on page 2-87). For register accesses in CR/CSR space, be sure to add 508 Kbytes (0x7F000) to the address offsets provided in the table.

The resets that apply to the registers are provided on a bit by bit basis. If "all" is given in the "Reset By" eld in the register description, then that bit is reset by power reset (PWRRST#), the VMEbus SYSRST* input (VRSYSRST#), and the PCI RST# input. If "PWR, VME" are given in the "Reset By" eld in the register description, then that bit is reset by power reset (PWRRST#) and the VMEbus SYSRST* input (VRSYSRST#). Bits not affected by resets contain "none" in the "Reset By" eld in the register description.

Table A.1 : Universe Register Map

Offset	Register Name	
000	PCI Conguration Space ID Register	PCI_ID
004	PCI Conguration Space Control and Status Register	PCI_CSR
008	PCI Conguration Class Register	PCI_CLASS
00C	PCI Conguration Miscellaneous 0 Register	PCI_MISC0
010	PCI Conguration Base Address Register	PCI_BS
014	PCI Unimplemented	
018	PCI Unimplemented	
01C	PCI Unimplemented	
020	PCI Unimplemented	
024	PCI Unimplemented	
028	PCI Reserved	
02C	PCI Reserved	
030	PCI Unimplemented	
034	PCI Reserved	
038	PCI Reserved	
03C	PCI Conguration Miscellaneous 1 Register	PCI_MISC1
040-0FF	PCI Unimplemented	
100	PCI Slave Image 0 Control	LSI0_CTL
104	PCI Slave Image 0 Base Address Register	LSI0_BS
108	PCI Slave Image 0 Bound Address Register	LSI0_BD
10C	PCI Slave Image 0 Translation Offset	LSI0_TO
110	Universe Reserved	

Table A.1 : Universe Register Map

Offset	Register Name	
114	PCI Slave Image 1 Control	LSI1_CTL
118	PCI Slave Image 1 Base Address Register	LSI1_BS
11C	PCI Slave Image 1 Bound Address Register	LSI1_BD
120	PCI Slave Image 1 Translation Offset	LSI1_TO
124	Universe Reserved	
128	PCI Slave Image 2 Control	LSI2_CTL
12C	PCI Slave Image 2 Base Address Register	LSI2_BS
130	PCI Slave Image 2 Bound Address Register	LSI2_BD
134	PCI Slave Image 2 Translation Offset	LSI2_TO
138	Universe Reserved	
13C	PCI Slave Image 3 Control	LSI3_CTL
140	PCI Slave Image 3 Base Address Register	LSI3_BS
144	PCI Slave Image 3 Bound Address Register	LSI3_BD
148	PCI Slave Image 3 Translation Offset	LSI3_TO
14C-16C	Universe reserved	
170	Special Cycle Control Register	SCYC_CTL
174	Special Cycle PCI bus Address Register	SCYC_ADDR
178	Special Cycle Swap/Compare Enable Register	SCYC_EN
17C	Special Cycle Compare Data Register	SCYC_CMP
180	Special Cycle Swap Data Register	SCYC_SWP
184	PCI Miscellaneous Register	LMISC
188	Special PCI Slave Image	SLSI
18C	PCI Command Error Log Register	L_CMDERR
190	PCI Address Error Log	LAERR
194-1FC	Universe reserved	
200	DMA Transfer Control Register	DCTL
204	DMA Transfer Byte Count Register	DTBC
208	DMA PCI bus Address Register	DLA
20C	Universe Reserved	
210	DMA VMEbus Address Register	DVA
214	Universe Reserved	
218	DMA Command Packet Pointer	DCPP
21C	Universe Reserved	
220	DMA General Control and Status Register	DGCS
224	DMA Linked List Update Enable Register	D_LLUE
228-2FC	Universe reserved	
300	PCI Interrupt Enable	LINT_EN

Table A.1 : Universe Register Map

Offset	Register Name	
304	PCI Interrupt Status	LINT_STAT
308	PCI Interrupt Map 0	LINT_MAP0
30C	PCI Interrupt Map 1	LINT_MAP1
310	VMEbus Interrupt Enable	VINT_EN
314	VMEbus Interrupt Status	VINT_STAT
318	VMEbus Interrupt Map 0	VINT_MAP0
31C	VMEbus Interrupt Map 1	VINT_MAP1
320	Interrupt Status/ID Out	STATID
324	VIRQ1 STATUS/ID	V1_STATID
328	VIRQ2 STATUS/ID	V2_STATID
32C	VIRQ3 STATUS/ID	V3_STATID
330	VIRQ4 STATUS/ID	V4_STATID
334	VIRQ5 STATUS/ID	V5_STATID
338	VIRQ6 STATUS/ID	V6_STATID
33C	VIRQ7 STATUS/ID	V7_STATID
340-3FC	Universe reserved	
400	Master Control	MAST_CTL
404	Miscellaneous Control	MISC_CTL
408	Miscellaneous Status	MISC_STAT
40C	User AM Codes Register	USER_AM
410-EFC	Universe reserved	
F00	VMEbus Slave Image 0 Control	VSI0_CTL
F04	VMEbus Slave Image 0 Base Address Register	VSI0_BS
F08	VMEbus Slave Image 0 Bound Address Register	VSI0_BD
F0C	VMEbus Slave Image 0 Translation Offset	VSI0_TO
F10	Universe reserved	
F14	VMEbus Slave Image 1 Control	VSI1_CTL
F18	VMEbus Slave Image 1 Base Address Register	VSI1_BS
F1C	VMEbus Slave Image 1 Bound Address Register	VSI1_BD
F20	VMEbus Slave Image 1 Translation Offset	VSI1_TO
F24	Universe reserved	
F28	VMEbus Slave Image 2 Control	VSI2_CTL
F2C	VMEbus Slave Image 2 Base Address Register	VSI2_BS
F30	VMEbus Slave Image 2 Bound Address Register	VSI2_BD
F34	VMEbus Slave Image 2 Translation Offset	VSI2_TO
F38	Universe reserved	
F3C	VMEbus Slave Image 3 Control	VSI3_CTL

Table A.1 : Universe Register Map

Offset	Register Name	
F40	VMEbus Slave Image 3 Base Address Register	VSI3_BS
F44	VMEbus Slave Image 3 Bound Address Register	VSI3_BD
F48	VMEbus Slave Image 3 Translation Offset	VSI3_TO
F4C-F6C	Universe reserved	
F70	VMEbus Register Access Image Control Register	VRAI_CTL
F74	VMEbus Register Access Image Base Address	VRAI_BS
F78	Universe reserved	
F7C	Universe reserved	
F80	VMEbus CSR Control Register	VCSR_CTL
F84	VMEbus CSR Translation Offset	VCSR_TO
F88	VMEbus AM Code Error Log	V_AMERR
F8C	VMEbus Address Error Log	VAERR
F90-FEC	Universe reserved	
FF0	VME CR/CSR Reserved	
FF4	VMEbus CSR Bit Clear Register	VCSR_CLR
FF8	VMEbus CSR Bit Set Register	VCSR_SET
FFC	VMEbus CSR Base Address Register	VCSR_BS



Caution: Register space marked as "reserved" should not be overwritten. Unimplemented registers return a value of 0 on reads; writes complete normally.

Table A.2 : PCI Conguration Space ID Register (PCI_ID)

Register Name: PCI_ID	Offset:h'00

Bits	Function
31-24	DID
23-16	DID
15-08	VID
07-00	VID

PCI_ID Description

Name	Type	Reset By	Reset State	Function
DID[15:0]	R	all	0	Device ID - Newbridge allocated device identier
VID[15:0]	R	all	h'0E3	Vendor ID - PCI SIG allocated vendor identier

Table A.3 : PCI Conguration Space Control and Status Register (PCI_CSR)

Register Name: PCI_CSR	Offset: 004

Bits	Function							
31-24	D_PE	S_SERR	R_MA	R_TA	S_TA	DEVSEL		DP_D
23-16	TFBBC	PCI Reserved						
15-08	PCI Reserved						MFBBC	SERR_EN
07-00	WAIT	PERESP	VGAPS	MWI_EN	SC	BM	MS	IOS

PCI_CSR Description

Name	Type	Reset By	Reset State	Function
D_PE	R/Write 1 to Clear	all	0	Detected Parity Error 0=No parity error, 1=Parity error This bit is always set by the Universe when: the PCI master interface detects a data parity error or the PCI target interface detects address or data parity errors.
S_SERR	R/Write 1 to Clear	all	0	Signalled SERR# 0=SERR# not asserted, 1=SERR# asserted. The Universe PCI target interface sets this bit when it asserts SERR# to signal an address parity error. SERR_EN must be set before SERR# can be asserted.
R_MA	R/Write 1 to Clear	all	0	Received Master-Abort 0=Master did not generate Master-Abort, 1=Master generated Master-Abort The Universe PCI master interface sets this bit when a transaction it initiated had to be terminated with a Master-Abort.
R_TA	R/Write 1 to Clear	all	0	Received Target-Abort 0=Master did not detect Target-Abort, 1=Master detected Target-Abort. The Universe PCI master interface sets this bit when a transaction it initiated was terminated with a Target-Abort.
S_TA	R/Write 1 to Clear	all	0	Signalled Target-Abort 0=Target did not terminate transaction with Target-Abort,1=Target terminated transaction with Target-Abort.
DEVSEL	R	all	01	Device Select Timing The Universe is a medium speed device
DP_D	R/Clear	all		Data Parity Detected 0=Master did not detect/generate data parity error, 1=Master detected/generated data parity error. The Universe PCI master interface sets this bit if the Parity Error Response bit is set, it is the master of transaction in which it asserts PERR#, or the addressed target asserts PERR#.
TFBBC	R	all	0	Target Fast Back to Back Capable Universe cannot accept Back to Back cycles from a different agent.

PCI_CSR Description

Name	Type	Reset By	Reset State	Function
MFBBC	R	all	0	Master Fast Back to Back Enable The Universe master never generates fast back to back transactions.
SERR_EN	R/W	all	0	SERR# Enable 0=Disable SERR# driver, 1=Enable SERR# driver. Setting this and PERESP allows the Universe PCI target interface to report address parity errors with SERR#.
WAIT	R	all	0	Wait Cycle Control 0=No address/data stepping
PERESP	R/W	all	0	Parity Error Response 0=Disable, 1=Enable Controls the Universe response to data and address parity errors. When enabled, it allows the assertion of PERR# to report data parity errors. When this bit and SERR_EN are asserted, the Universe can report address parity errors on SERR#. Universe parity generation is unaffected by this bit.
VGAPS	R	all	0	VGA Palette Snoop 0=Disable
MWI_EN	R	all	0	Memory Write and Invalidate Enable 0=Disable The Universe PCI master interface never generates a Memory Write and Invalidate command.
SC	R	all	0	Special Cycles 0=Disable The Universe PCI target interface never responds to special cycles.
BM	R/W	PWR, VME	see note 1	Master Enable 0=Disable, 1=Enable For a VMEbus slave image to respond to an incoming cycle, this bit must be set.
MS	R/W	PWR, VME	see note 1	Target Memory Enable 0=Disable, 1=Enable
IOS	R/W	PWR, VME	see note 1	Target IO Enable 0=Disable, 1=Enable

Note:

1. These bits may not be disabled after reset as required by the PCI specification if the VCSR or LSI0 power-up options have been enabled.

Table A.4 : PCI Conguration Class Register (PCI_CLASS)

Register Name: PCI_CLASS	Offset:08H

Bits	Function
31-24	BASE [7:0]
23-16	SUB [7:0]
15-08	PROG [7:0]
07-00	RID [7:0]

PCI_CLASS Description

Name	Type	Reset By	Reset State	Function
BASE [7:0]	R	all	06	Base Class Code The Universe is dened as a PCI bridge device
SUB [7:0]	R	all	80	Sub Class Code The universe sub-class is "other bridge device"
PROG [7:0]	R	all	00	Programming Interface The Universe does not have a standardized register-level programming interface
RID [7:0]	R	all	00	Revision ID

Table A.5 : PCI Conguration Miscellaneous 0 Register (PCI_MISC0)

Register Name: PCI_MISC0						Offset: 00C
Bits	**Function**					
31-24	BISTC	SBIST	PCI Reserved	CCODE		
23-16	MFUNCT	LAYOUT				
15-08	LTIMER			0	0	0
07-00	PCI Unimplemented					

PCI_MISC0 Description

Name	Type	Reset By	Reset State	Function
BISTC	R	all	0	The Universe is not BIST Capable
SBIST	R	all	0	Start BIST The Universe is not BIST capable
CCODE	R	all	0	Completion Code The Universe is not BIST capable
MFUNCT	R	all	1	Multifunction Device 0=No, 1=Yes The Universe is a multifunction device with a single function: function 0.
LAYOUT	R	all	0	Conguration Space Layout
LTIMER [7:3]	R/W	all	0	Latency Timer: The latency timer has a resolution of 8 clocks

Table A.6 : PCI Conguration Base Address Register (PCI_BS)

Register Name: PCI_BS	Offset: 038

Bits	Function							
31-24	BS							
23-16	BS							
15-08	0	0	0	0	0	0	0	0
07-00	0	0	0	0	0	0	0	SPACE

PCI_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	all	0	Base Address
SPACE	R	all	Power-up Option	PCI bus Address Space 0=Memory, 1=I/O

This register species the 64 Kbyte aligned base address of the Universe register space on PCI. Since the Universe register space is only 4 Kbytes, it only responds to this image if address lines [15:12] are zero. PCI address lines [11:0] are used to select the Universe register.

A power-up option determines whether the registers are mapped into Memory or I/O space. There is an internal inversion on the power-up pin (VA[1]). When VA[1] pin is pulled down at power-up, the device starts-up in I/O space and the SPACE bit is set to 1. When VA[1] is pulled up at power-up, the Universe starts-up in Memory space. If mapped into Memory space, the user is free to locate the registers anywhere in the 32-bit address space, but note that the registers are not prefetchable.

Since all Memory and I/O accesses are specied in relation to the Base Address, a write must occur to this register before any Universe registers can be accessed through PCI Memory or I/O space. This initial write can be performed with a PCI conguration transaction or a VMEbus register access.

Table A.7 : PCI Conguration Miscellaneous 1 Register (PCI_MISC1)

Register Name: PCI_MISC1	Offset:03C

Bits	Function
31-24	MAX_LAT [7:0]
23-16	MIN_GNT [7:0]
15-08	INT_PIN [7:0]
07-00	INT_LINE [7:0]

PCI_MISC1 Description

Name	Type	Reset By	Reset State	Function
MAX_LAT[7:0]	R	all	0	Maximum Latency: This device has no special latency requirements
MIN_GNT[7:0]	R	all	00000011	Minimum Grant: 250 ns units
INT_PIN[7:0]	R	all	00000001	Interrupt Pin: Universe pin INT# [0] has a PCI compliant I/O buffer
INT_LINE[7:0]	R/W	all	0	Interrupt Line: used by some PCI systems to record interrupt routing information

The MIN_GNT parameter assumes the Universe master is transferring an aligned burst size of 64 bytes to a 32-bit target with no wait states. This would require roughly 20 clocks (at a clock frequency of 33 MHz, this is about 600 ns). MIN_GNT is set to three, or 750 ns.

Table A.8 : PCI Slave Image 0 Control (LSI0_CTL)

Register Name: LSI0_CTL																																OFFSET: 100

Bits	Function																															
31-24	EN		PWEN						Universe Reserved																							
23-16	VDW								Universe Reserved																VAS							
15-08	PGM								SUPER								Universe Reserved														VCT	
07-00	Universe Reserved																														LAS	

LSI0_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	Power-up Option	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
VDW	R/W	all	10	VMEbus Maximum Datawidth 00=8 bit data width, 01=16 bit data width, 10=32 bit data width, 11=64 bit data width
VAS	R/W	all	Power-up Option	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved 101=CR/CSR, 110=User1, 111=User2
PGM	R/W	all	0	Program/Data AM Code 00=Data, 01=Program, others=Reserved
SUPER	R/W	all	0	Supervisor/User AM Code 00=Non-Privileged, 01=Supervisor, others=Reserved
VCT	R/W	all	0	VMEbus Cycle Type 0=Single Cycles Only on VMEbus, 1=Single Cycles and Block Transfers on VMEbus
LAS	R/W	all	Power-up Option	PCI bus Memory Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Type 1 Conguration Space, 11=Reserved

The general, VMEbus and PCI bus controls for this A32 capable image. This PCI slave image has 4 KByte granularity, and dual address cycles are never claimed. Only PCI Memory write transactions can be posted.

Table A.9 : PCI Slave Image 0 Base Address Register (LSI0_BS)

Register Name: LSI0_BS	Offset: 104

Bits	Function	
31-24	BS	
23-16	BS	
15-08	BS	Universe Reserved
07-00	Universe Reserved	

LSI0_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:28]	R/W	all	Power-up Option	Base Address
BS[27:12]	R/W	all	0	Base Address

The base address species the lowest address in the address range that will be decoded.

Table A.10 : PCI Slave Image 0 Bound Address Register (LSI0_BD)

Register Name: LSI0_BD	Offset:10C

Bits	Function	
31-24	BD	
23-16	BD	
15-08	BD	Universe Reserved
07-00	Universe Reserved	

LSI0_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:28]	R/W	all	Power-up Option	Bound Address
BD[27:12]	R/W	all	0	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register. If the bound address is 0, then the addresses decoded are those greater than or equal to the base address.

Table A.11 : PCI Slave Image 0 Translation Offset (LSI0_TO)

Register Name: LSI0_TO	Offset: 10C

Bits	Function	
31-24	TO	
23-16	TO	
15-08	TO	Universe Reserved
07-00	Universe Reserved	

LSI0_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:12]	R/W	all	0	Translation Offset

The TO is eld is used in an unsigned addition with PCIBS and the PCI address to generate a VME address. For example, if PCIBS is 32'h0000_0000 and TO is 32'h8000_0000 then the corresponding VME address will be equal to the addition of AD[11:0] and 32'h8000_0000.

Table A.12 : PCI Slave Image 1 Control (LSI1_CTL)

Register Name: LSI1_CTL	Offset: 114

Bits	Function					
31-24	EN	PWEN	Universe Reserved			
23-16	VDW		Universe Reserved		VAS	
15-08	PGM		SUPER	Universe Reserved		VCT
07-00	Universe Reserved				LAS	

LSI1_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
VDW	R/W	all	10	VMEbus Maximum Datawidth 00=8 bit data width, 01=16 bit data width, 10=32 bit data width, 11=64 bit data width
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=CR/CSR, 110=User1, 111=User2
PGM	R/W	all	0	Program/Data AM Code 00=Data, 01=Program, others=Reserved
SUPER	R/W	all	0	Supervisor/User AM Code 00=Non-Privileged, 01=Supervisor, others=Reserved
VCT	R/W	all	0	VMEbus Cycle Type 0=Single Cycles Only on VMEbus, 1=Single Cycles and Block Transfers on VMEbus
LAS	R/W	all	0	PCI bus Memory Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Type 1 Conguration Space, 11=Reserved

The general, VMEbus and PCI bus controls for this A32 capable image. This PCI slave image has 64 KByte granularity (maximum size of 4GBytes), and dual address cycles are never claimed. Only PCI Memory write transactions can be posted.

Table A.13 : PCI Slave Image 1 Base Address Register (LSI1_BS)

Register Name: LSI1_BS	Offset: 118

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

LSI1_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.14 : PCI Slave Image 1 Bound Address Register (LSI1_BD)

Register Name: LSI1_BD	Offset:11C

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

LSI1_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register.

Table A.15 : PCI Slave Image 1 Translation Offset (LSI1_TO)

Register Name: LSI1_TO	Offset:120

Bits	Function
31-24	TO
23-16	TO
15-08	Universe Reserved
07-00	Universe Reserved

LSI1_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:16]	R/W	none	x	Translation offset

Table A.16 : PCI Slave Image 2 Control (LSI2_CTL)

Register Name: LSI2_CTL								Offset=128
Bits	**Function**							
31-24	EN	PWEN	Universe Reserved					
23-16	VDW		Universe Reserved			VAS		
15-08	PGM		SUPER		Universe Reserved			VCT
07-00	Universe Reserved						LAS	

LSI2_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
VDW	R/W	all	10	VMEbus Maximum Datawidth 00=8 bit data width, 01=16 bit data width, 10=32 bit data width, 11=64 bit data width
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=CR/CSR, 110=User1, 111=User2
PGM	R/W	all	0	Program/Data AM Code 00=Data, 01=Program, others=Reserved
SUPER	R/W	all	0	Supervisor/User AM Code 00=Non-Privileged, 01=Supervisor, others=Reserved
VCT	R/W	all	0	VMEbus Cycle Type 0=Single Cycles Only on VMEbus, 1=Single Cycles and Block Transfers on VMEbus
LAS	R/W	all	0	PCI bus Memory Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Type 1 Conguration Space, 11=Reserved

The general, VMEbus and PCI bus controls for this A32 capable image. This PCI slave image has 64 KByte granularity, and dual address cycles are never claimed. Only PCI Memory write transactions can be posted.

Table A.17 : PCI Slave Image 2 Base Address Register (LSI2_BS)

Register Name: LSI2_BS	Offset:12C

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

LSI2_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.18 : PCI Slave Image 2 Bound Address Register (LSI2_BD)

Register Name: LSI2_BD	Offset:130

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

LSI2_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register.

Table A.19 : PCI Slave Image 2 Translation Offset (LSI2_TO)

Register Name: LSI2_TO	Offset:134

Bits	Function
31-24	TO
23-16	TO
15-08	Universe Reserved
07-00	Universe Reserved

LSI2_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:16]	R/W	none	x	Translation offset

Table A.20 : PCI Slave Image 3 Control (LSI3_CTL)

Register Name: LSI3_CTL								Offset: 13C	

Bits	Function								
31-24	EN	PWEN	Universe Reserved						
23-16	VDW		Universe Reserved				VAS		
15-08	PGM		SUPER		Universe Reserved				VCT
07-00	Universe Reserved							LAS	

LSI3_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
VDW	R/W	all	10	VMEbus Maximum Datawidth 00=8 bit data width, 01=16 bit data width, 10=32 bit data width, 11=64 bit data width
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=CR/CSR, 110=User1, 111=User2
PGM	R/W	all	0	Program/Data AM Code 00=Data, 01=Program, others=Reserved
SUPER	R/W	all	0	Supervisor/User AM Code 00=Non-Privileged, 01=Supervisor, others=Reserved
VCT	R/W	all	0	VMEbus Cycle Type 0=Single Cycles Only on VMEbus, 1=Single Cycles and Block Transfers on VMEbus
LAS	R/W	all	0	PCI bus Memory Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Type 1 Conguration Space, 11=Reserved

The general, VMEbus and PCI bus controls for this A32 capable image. This PCI slave image has 64 KByte granularity (maximum size of 4GBytes), and dual address cycles are never claimed. Only PCI Memory write transactions can be posted.

Table A.21 : PCI Slave Image 3 Base Address Register (LSI3_BS)

Register Name: LSI3_BS	Offset:140

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

LSI3_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.22 : PCI Slave Image 3 Bound Address Register (LSI3_BD)

Register Name: LSI3_BD	Offset:144

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

LSI3_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register.

Table A.23 : PCI Slave Image 3 Translation Offset (LSI3_TO)

Register Name: LSI3_TO	Offset:148

Bits	Function
31-24	TO
23-16	TO
15-08	Universe Reserved
07-00	Universe Reserved

LSI3_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:16]	R/W	none	x	Translation offset

Table A.24 : Special Cycle Control Register (SCYC_CTL)

Register Name: SCYC_CTL	Offset: 170

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	
07-00	Universe Reserved	SCYC

SCYC_CTL Description

Name	Type	Reset By	Reset State	Function
SCYC	R/W	all	0	Special Cycle 00=Disable, 01=RMW, 10=ADOH, 11=Reserved

The special cycle generator will generate an ADOH or RMW cycle for the 32-bit PCI bus address which matches the programmed address in SCYC_ADDR. A Read-Modify-Write command is initiated by a read to the specied address. Address-Only cycles are initiated by either read or write cycles.

Table A.25 : Special Cycle PCI bus Address Register

Register Name: SCYC_ADDR	Offset: 174

Bits	Function	
31-24	ADDR	
23-16	ADDR	
15-08	ADDR	
07-00	ADDR	Universe Reserved

SCYC_ADDR Description

Name	Type	Reset By	Reset State	Function
ADDR[31:2]	R/W	none	x	Address

This register designates the special cycle address. This address must be contained in an enabled PCI image.

Table A.26 : Special Cycle Swap/Compare Enable Register (SCYC_EN)

Register Name: SCYC_EN	Offset: 278

Bits	Function
31-24	EN
23-16	EN
15-08	EN
07-00	EN

SCYC_EN Description

Name	Type	Reset By	Reset State	Function
EN[31:0]	R/W	none	x	Bit Enable 0=Disable, 1=Enable

The bits enabled in this register denes the bits that will be involved in the compare and swap operations for VME RMW cycles.

Table A.27 : Special Cycle Compare Data Register (SCYC_CMP)

Register Name: SCYC_CMP	Offset: 17C

Bits	Function
31-24	CMP
23-16	CMP
15-08	CMP
07-00	CMP

SCYC_CMP Description

Name	Type	Reset By	Reset State	Function
CMP[31:0]	R/W	none	x	The data returned from the VMEbus is compared with the contents of this register.

The data returned from the read portion of a VMEbus RMW is compared with the contents of this register. SCYC_EN is used to control which bits are compared.

Table A.28 : Special Cycle Swap Data Register (SCYC_SWP)

Register Name: SCYC_SWP	Offset: 180

Bits	Function
31-24	SWP
23-16	SWP
15-08	SWP
07-00	SWP

SCYC_SWP Description

Name	Type	Reset By	Reset State	Function
SWP[31:0]	R/W	none	x	Swap data

If enabled bits matched with the value in the compare register, then the contents of the swap data register is written back to VME. SCYC_EN is used to control which bits are written back to VME.

Table A.29 : PCI Miscellaneous Register (LMISC)

Register Name: LMISC			Offset: 104

Bits	Function	
31-24	CRT	CWT
23-16	Universe Reserved	
15-08	Universe Reserved	
07-00	Universe Reserved	

LMISC Description

Name	Type	Reset By	Reset State	Function
CRT	R/W	all	0001	Coupled Request Timer 0000=Disable (Innite), 0001=128 µsec, 0010=256 µsec, 0011=512 µsec, 0100=1024 µsec, 0101=2048 µsec, 0110=4096 µsec, others=RESERVED
CWT	R/W	all	0	Coupled Window Timer 0000=Disable - release after rst coupled transaction, 0001=16 PCI Clocks, 0010=32 PCI Clocks, 0011=64 PCI Clocks, 0100=128 PCI Clocks, 0101=256 PCI Clocks, 0110=512 PCI Clocks, others=RESERVED

If a PCI master generates a coupled transaction and the PCI Slave Channel in the Universe is not currently the owner of the VMEbus, then the Universe retries the PCI master and requests VMEbus ownership. The Universe uses the coupled request timer (CRT) to determine how long to wait after ownership is acquired for a PCI master to retry a coupled transaction. The timer is restarted each time a PCI master attempts the coupled transaction. If this timer expires, then the PCI Slave Channel releases the VME Master Interface.

The Universe uses the coupled window timer (CWT) to determine how long to hold ownership of the VMEbus on behalf of the PCI Slave Channel after processing a coupled transaction. The timer is restarted each time the Universe processes a coupled transaction. If this timer expires, then the PCI Slave Channel releases the VME Master Interface.

Device behaviour is unpredictable if CRT/CWT are changed during coupled cycle activity. These registers should be set only at conguration time, or after disabling all PCI Slave Images.

Table A.30 : Special PCI Slave Image (SLSI)

Register Name: SLSI	Offset: 18C

Bits	Function			
31-24	EN	PWEN	Universe Reserved	
23-16	VDW		Universe Reserved	
15-08	PGM		SUPER	
07-00	BS			LAS

SLSI Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
VDW [3:0]	R/W	all	0	VMEbus Maximum Datawidth Each of the four bits species a data width for the corresponding 16 MByte region. Low order bits correspond to the lower address regions. 0=16-bit, 1=32-bit
PGM [3:0]	R/W	all	0	Program/Data AM Code Each of the four bits species Program/Data AM code for the corresponding 16 MByte region. Low order bits correspond to the lower address regions. 0=Data, 1=Program
SUPER [3:0]	R/W	all	0	Supervisor/User AM Code Each of the four bits species Supervisor/User AM code for the corresponding 16 MByte region. Low order bits correspond to the lower address regions. 0=Non-Privileged, 1=Supervisor
BS [5:0]	R/W	all	0	Base Address Species a 64 MByte aligned base address for this 64 MByte image.
LAS	R/W	all	0	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Type 1 Conguration Space, 11=Reserved

This register fully species an A32 capable special PCI slave image. The base is programmable to a 64 Mbyte alignment, and the size is xed at 64 Mbytes. Incoming address lines [31:26] (in Memory, I/O, or Conguration space) must match this eld for the Universe to decode the access. This special PCI slave image has lower priority than any other PCI slave image.

The 64 Mbytes of the SLSI is partitioned into four 16 Mbyte regions, numbered 0 to 3 (0 is at the lowest address). PCI address bits [25:24] are used to select regions. The top 64 Kbyte of each region is mapped to VMEbus A16 space, and the rest of each 16 Mbyte region is mapped to A24 space.

The user can use the PGM, SUPER and VDW elds to specify the AM code and the maximum port size for each region. The PGM eld is ignored for the portion of each region mapped to A16 space.

No block transfer AM codes are generated.

Table A.31 : PCI Command Error Log Register (L_CMDERR)

Register Name: L_CMDERR	Offset: 13C

Bits	Function			
31-24	CMDERR		M_ERR	Universe Reserved
23-16	L_STAT	Universe Reserved		
15-08	Universe Reserved			
07-00	Universe Reserved			

L_CMDERR Description

Name	Type	Reset By	Reset State	Function
CMDERR [3:0]	R	all	0111	PCI Command Error Log
M_ERR	R	all	0	Multiple Error Occurred 0=Single error, 1=At least one error has occurred since the logs were frozen.
L_STAT	R/W	all	0	PCI Error Log Status Reads: 0=logs invalid, 1=logs are valid and error logging halted Writes: 0=no effect, 1=clears L_STAT and enables error logging

The Universe PCI Master Interface is responsible for logging errors under the following conditions:

- a posted write transaction results in a target abort,
- a posted write transaction results in a master abort, or
- a maximum retry counter expires during retry of posted write transaction.

This register logs the command information.

Table A.32 : PCI Address Error Log (LAERR)

Register Name: LAERR	Offset: 190

Bits	Function
31-24	LAERR
23-16	LAERR
15-08	LAERR
07-00	LAERR

LAERR Description

Name	Type	Reset By	Reset State	Function
LAERR [31:0]	R	all	0	PCI lower address error log

The starting address of an errored PCI transaction is logged in this register under the following conditions:

- a posted write transaction results in a target abort,
- a posted write transaction results in a master abort, or
- a maximum retry counter expires during retry of posted write transaction.

Contents are qualied by bit L_STAT of the L_CMDERR register.

Table A.33 : DMA Transfer Control Register (DCTL)

Register Name: DCTL	Offset: 200

Bits	Function				
31-24	L2V	Universe Reserved			
23-16	VDW	Universe Reserved		VAS	
15-08	PGM	SUPER	Universe Reserved		VCT
07-00	LD64EN	Universe Reserved			

DCTL Description

Name	Type	Reset By	Reset State	Function
L2V	R/W	all	0	Direction 0=Transfer from VMEbus to PCI Bus, 1=Transfer from PCI bus to VMEbus
VDW	R/W	all	0	VMEbus Maximum Datawidth 00=8 bit data width, 01=16 bit data width, 10=32 bit data width, 11=64 bit data width
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=Reserved, 110=User1, 111=User2
PGM	R/W	all	0	Program/Data AM Code 00=Data, 01=Program, others=Reserved
SUPER	R/W	all	0	Supervisor/User AM Code 00=Non-Privileged, 01=Supervisor, others=Reserved
VCT	R/W	all	0	VMEbus Cycle Type 0=Single Cycles Only on VMEbus, 1=Single Cycles and Block Transfers on VMEbus
LD64EN	R/W	all	1	Enable 64-bit PCI Bus Transactions 0=Disable, 1=Enable

This register is programmed from either bus or is programmed by the DMAC when it loads the command packet. The DMA only accesses PCI bus Memory space.

Table A.34 : DMA Transfer Byte Count Register (DTBC)

Register Name: DTBC		Offset: 204

Bits	Function
31-24	Universe Reserved
23-16	DTBC
15-08	DTBC
07-00	DTBC

DTBC Description

Name	Type	Reset By	Reset State	Function
DTBC[23:0]	R/W	all	0	DMA Transfer Byte Count

This register species the number of bytes to be moved by the DMA before the start of the DMA transfer, or the number of remaining bytes in the transfer while the DMA is active. This register is programmed from either bus or is programmed by the DMAC when it loads the command packet. If the register is non-zero at the start of a linked-list operation, these bytes are transferred before reading in the rst linked-list entry.

Table A.35 : DMA PCI Bus Address Register (DLA)

Register Name: DLA	Offset: 0x214

Bits	Function
31-24	LA
23-16	LA
15-08	LA
07-00	LA

DLA Description

Name	Type	Reset By	Reset State	Function
LA[31:3]	R/W	none	x	PCI bus Address
LA[2:0]	R/W	all	0	PCI bus Address

This register is programmed from either bus or is programmed by the DMAC when it loads the command packet. During linked-list operation, this register is only updated at the end of a transaction (i.e., processing of a command packet) or when the DMA is stopped or halted.

Table A.36 : DMA VMEbus Address Register (DVA)

Register Name: DVA				Offset: 27C

Bits	Function
31-24	VA
23-16	VA
15-08	VA
07-00	VA

DVA Description

Name	Type	Reset By	Reset State	Function
VA[31:3]	R/W	none	x	VMEbus Address
VA[2:0]	R/W	all	0	VMEbus Address

This register is programmed from either bus or is programmed by the DMAC when it loads the command packet. During linked-list operation, this register is only updated at the end of a transaction (i.e., processing of a command packet) or when the DMA is stopped or halted.

Table A.37 : DMA Command Packet Pointer (DCPP)

Register Name: DCPP	Offset:218

Bits	Function	
31-24	DCPP	
23-16	DCPP	
15-08	DCPP	
07-00	DCPP	Universe Reserved

DCPP Description

Name	Type	Reset By	Reset State	Function
DCPP[31:3]	R/W	all	0	DMA Command Packet Pointer

This register contains the pointer into the current command packet. Initially it is programmed to the starting packet of the linked-list, and is updated with the address to a new command packet at the completion of a packet. The packets must be at an aligned 64-bit address.

Table A.38 : DMA General Control/Status Register (DGCS)

Register Name: DGCS							Offset: 22H	

Bits	Function							
31-24	GO	STOP_REQ	HALT_REQ	0	CHAIN	0	0	0
23-16	VON				VOFF			
15-08	ACT	STOP	HALT	0	DONE	LERR	VERR	P_ERR
07-00	0	INT_STOP	INT_HALT	0	INT_DONE	INT_LERR	INT_VERR	INT_P_ERR

DGCS Description

Name	Type	Reset By	Reset State	Function
GO	W/Read 0 always	all	0	DMA Go Bit 0=No effect, 1=Enable DMA Transfers
STOP_REQ	W/Read 0 always	all	0	DMA Stop Request 0=No effect, 1=Stop DMA transfer when all buffered data has been written
HALT_REQ	W/Read 0 always	all	0	DMA Halt Request 0=No effect, 1=Halt the DMA transfer at the completion of the current command packet
CHAIN	R/W	all	0	DMA Chaining 0=DMA Direct Mode, 1=DMA Linked List mode
VON [3:0]	R/W	all	0	VMEbus Aligned DMA Transfer Count 0000=Until done, 0001=256 bytes, 0010=512 bytes, 0011=1024 bytes, 0100=2048 bytes, 0101=4096 bytes, 0110=8192 bytes, 0111=16384 bytes, others=Reserved
VOFF [3:0]	R/W	all	0	Minimum period the DMA is off the VMEbus between tenures 0000=0µs, 0001=16µs, 0010=32µs, 0011=64µs, 0100=128µs, 0101=256µs, 0110=512µs, 0111=1024µs, others=Reserved
ACT	R	all	0	DMA Active Flag 0=Not Active, 1=Active
STOP	R/Write 1 to Clear	all	0	DMA Stopped Flag 0=Not Stopped, 1=Stopped
HALT	R/Write 1 to Clear	all	0	DMA Halted Flag 0=Not Halted, 1=Halted
DONE	R/Write 1 to Clear	all	0	All DMA Transfers Complete 0=Not Complete, 1=Complete
LERR	R/Write 1 to Clear	all	0	DMA PCI Bus Error 0=No Error, 1=Error
VERR	R/Write 1 to Clear	all	0	DMA VMEbus Error 0=No Error, 1=Error
P_ERR	R/Write 1 to Clear	all	0	Protocol Error Asserted if PCI master interface disabled or lower three bits of PCI and VME addresses differ 0=No Error, 1=Error

DGCS Description

Name	Type	Reset By	Reset State	Function
INT_STOP	R/W	all	0	Interrupt when Stopped 0=Disable, 1=Enable
INT_HALT	R/W	all	0	Interrupt when Halted 0=Disable, 1=Enable
INT_DONE	R/W	all	0	Interrupt when Done 0=Disable, 1=Enable
INT_LERR	R/W	all	0	Interrupt on LERR 0=Disable, 1=Enable
INT_VERR	R/W	all	0	Interrupt on VERR 0=Disable, 1=Enable
INT_P_ERR	R/W	all	0	Interrupt on Protocol Error 0=Disable, 1=Enable

STOP, HALT, DONE, LERR, VERR, and P_ERR all need to be cleared to enable the GO bit to start DMA transfers.

Table A.39 : DMA Linked List Update Enable Register (D_LLUE)

Register Name: D_LLUE	Offset:224

Bits	Function	
31-24	UPDATE	Universe Reserved
23-16	Universe Reserved	
15-08	Universe Reserved	
07-00	Universe Reserved	

D_LLUE Description

Name	Type	Reset By	Reset State	Function
UPDATE	R/W	all	0	DMA Linked List Update Enable 0=PCI Resource not Updating Linked List 1=PCI Resource Updating Linked List

The PCI Resource must read back a logic 1 in the UPDATE eld before proceeding to modify the linked list. After the Linked List has been modied the PCI Resource must clear the UPDATE eld by writing a logic 0.

Table A.40 : PCI Interrupt Enable Register (LINT_EN)

Register Name: LINT_EN								Offset: 300
Bits	**Function**							
31-24	Universe Reserved							
23-16								
15-08	ACFAIL	SYSFAIL	SW_INT	SW_IACK	Reserved	VERR	LERR	DMA
07-00	VIRQ7	VIRQ6	VIRQ5	VIRQ4	VIRQ3	VIRQ2	VIRQ1	VOWN

LINT_EN Description

Name	Type	Reset By	Reset State	Function
ACFAIL	R/W	all	0	ACFAIL Interrupt Mask 0=ACFAIL Interrupt masked 1=ACFAIL Interrupt enabled
SYSFAIL	R/W	all	0	SYSFAIL Interrupt Mask 0=SYSFAIL Interrupt masked 1=SYSFAIL Interrupt enabled
SW_INT	R/W	all	0	PCI Software Interrupt Mask 0=PCI Software Interrupt masked 1=PCI Software Interrupt enabled A zero-to-one transition will cause the PCI software interrupt to be asserted. Subsequent zeroing of this bit will cause the interrupt to be masked, but will not clear the PCI Software Interrupt Status bit.
SW_IACK	R/W	all	0	"VME Software IACK" Mask 0 ="VME Software IACK" Interrupt masked 1 ="VME Software IACK" Interrupt enabled
VERR	R/W	all	0	PCI VERR Interrupt Mask 0 =PCI VERR Interrupt masked 1=PCI VERR Interrupt enabled
LERR	R/W	all	0	PCI LERR Interrupt Mask 0 =PCI LERR Interrupt masked 1 =PCI LERR Interrupt enabled
DMA	R/W	all	0	PCI DMA Interrupt Mask 0=PCI DMA Interrupt masked 1=PCI DMA Interrupt enabled
VIRQ7-VIRQ1	R/W	all	0	VIRQ*x* Interrupt Mask 0=VIRQ*x* Interrupt masked 1 =VIRQ*x* Interrupt enabled
VOWN	R/W	all	0	VOWN Interrupt Mask 0=VOWN Interrupt masked 1=VOWN Interrupt Enabled

Bits VIRQ7-VIRQ1 enable the Universe to respond as a VME Interrupt Handler to interrupts on the VIRQ[x] lines. When a VIRQx interrupt is enabled, and the corresponding VIRQ[x] pin is asserted, the Universe requests the VMEbus and performs a VME IACK cycle for that interrupt level. When the interrupt acknowledge cycle completes, the STATUS/ID is stored in the corresponding VINT_ID register, the VIRQx bit of the LINT_STAT register is set, and a PCI interrupt is generated. The Universe does not acquire further interrupt STATUS/ID vectors at the same interrupt level until the VIRQx bit in the LINT_STAT register is cleared.

The other bits enable the respective internal or external sources to interrupt the PCI side.

Table A.41 : PCI Interrupt Status Register (LINT_STAT)

Register Name: LINT_STAT								Offset: 304
Bits	**Function**							
31-24	Universe Reserved							
23-16								
15-08	ACFAIL	SYSFAIL	SW_INT	SW_IACK	Reserved	VERR	LERR	DMA
07-00	VIRQ7	VIRQ6	VIRQ5	VIRQ4	VIRQ3	VIRQ2	VIRQ1	VOWN

LINT_STAT Description

Name	Type	Reset By	Reset State	Function
ACFAIL	R	all	0	ACFAIL Interrupt Status/Clear 0: no ACFAIL Interrupt 1: ACFAIL Interrupt active
SYSFAIL	R	all	0	SYSFAIL Interrupt Status/Clear 0: no SYSFAIL Interrupt 1: SYSFAIL Interrupt active
SW_INT	R/Write 1 to Clear	all	0	PCI Software Interrupt Status/Clear 0: no PCI Software Interrupt 1: PCI Software Interrupt active
SW_IACK	R/Write 1 to Clear	all	0	"VME Software IACK" Status/Clear 0: no "VME Software IACK" Interrupt 1: "VME Software IACK" Interrupt active
VERR	R/Write 1 to Clear	all	0	PCI VERR Interrupt Status/Clear 0: no PCI VERR Interrupt 1: PCI VERR Interrupt active
LERR	R/Write 1 to Clear	all	0	PCI LERR Interrupt Status/Clear 0: no PCI LERR Interrupt 1: PCI LERR Interrupt active
DMA	R/Write 1 to Clear	all	0	PCI DMA Interrupt Status/Clear 0: no PCI DMA Interrupt 1: PCI DMA Interrupt active
VIRQ7-VIRQ1	R/Write 1 to Clear	all	0	VIRQx Interrupt Status/Clear 0: no VIRQx Interrupt StatusID vector available 1: VIRQx Interrupt StatusID vector available
VOWN	R/Write 1 to Clear	all	0	VOWN Interrupt Status/Clear 0: no VOWN Interrupt active 1: VOWN Interrupt active

Interrupt status bits indicated as "R" (read only) are level sensitive, and the source of the interrupt must be cleared in order to clear the interrupt status. ACFAIL and SYSFAIL bits directly reect the status of the respective VMEbus pins (after ltering and synchronization to the PCI clock).

Status bits indicated as "R/Write 1 to Clear" are edge sensitive: the status is latched when the interrupt event occurs. These status bits can be cleared independently of the state of the interrupt source by writing a "1" to the status register. Clearing the status bit does not imply the source of the interrupt is cleared.

Table A.42 : PCI Interrupt Map 0 Register (LINT_MAP0)

Register Name: LINT_MAP0				Offset: 400

Bits	Function			
31-24	Reserved	VIRQ7	Reserved	VIRQ6
23-16	Reserved	VIRQ5	Reserved	VIRQ4
15-08	Reserved	VIRQ3	Reserved	VIRQ2
07-00	Reserved	VIRQ1	Reserved	VOWN

LINT_MAP0 Description

Name	Type	Reset By	Reset State	Function
VIRQ7-VIRQ1	R/W	all	0	PCI interrupt destination (LINT[7:0]) for VIRQx
VOWN	R/W	all	0	VMEbus ownership bit interrupt map to PCI interrupt

The two PCI Interrupt Map registers map various interrupt sources to one of the eight PCI interrupt pins. A value of 000 maps the corresponding interrupt source to LINT# [0], a value of 001 maps to LINT# [1], etc.

Table A.43 : PCI Interrupt Map 1 Register (LINT_MAP1)

Register Name: LINT_MAP1				Offset: 30C

Bits	Function			
31-24	Reserved	ACFAIL	Reserved	SYSFAIL
23-16	Reserved	SW_INT	Reserved	SW_IACK
15-08	Reserved			VERR
07-00	Reserved	LERR	Reserved	DMA

LINT_MAP1 Description

Name	Type	Reset By	Reset State	Function
ACFAIL	R/W	all	0	ACFAIL interrupt destination
SYSFAIL	R/W	all	0	SYSFAIL interrupt destination
SW_INT	R/W	all	0	PCI software interrupt destination
SW_IACK	R/W	all	0	VMEbus Software IACK interrupt destination
VERR	R/W	all	0	VMEbus Error interrupt destination
LERR	R/W	all	0	PCI bus Error interrupt destination
DMA	R/W	all	0	DMA interrupt destination

The two PCI Interrupt Map registers map various interrupt sources to one of the eight PCI interrupt pins. A value of 000 maps the corresponding interrupt source to LINT# [0], a value of 001 maps to LINT# [1], etc.

Table A.44 : VMEbus Interrupt Enable Register (VINT_EN)

Register Name: VINT_EN								Offset: 210
Bits	**Function**							
31-24	Universe Reserved							
23-16								
15-08	Reserved			SW_INT	Reserved	VERR	LERR	DMA
07-00	LINT7	LINT6	LINT5	LINT4	LINT3	LINT2	LINT1	LINT0

VINT_EN Description

Name	Type	Reset By	Reset State	Function
SW_INT	R/W	all	Power-up Option	"VME Software Interrupt" Mask 0 = VME Software Interrupt masked 1 =VME Software Interrupt enabled A zero-to-one transition causes the VME software interrupt to be asserted. Subsequent zeroing of this bit causes the interrupt to be masked and the VMEbus interrupt negated, but does not clear the VME software interrupt status bit.
VERR	R/W	all	0	VERR Interrupt Mask 0 =PCI VERR Interrupt masked 1=PCI VERR Interrupt enabled
LERR	R/W	all	0	LERR Interrupt Mask 0 =PCI LERR Interrupt masked 1 =PCI LERR Interrupt enabled
DMA	R/W	all	0	DMA Interrupt Mask 0=PCI DMA Interrupt masked 1=PCI DMA Interrupt enabled
LINT7-LINT0	R/W	all	0	PCI Interrupt Mask 0=LINTx Interrupt masked 1 =LINTx Interrupt enabled

This register enables the various sources of VMEbus interrupts. SW_INT can be enabled with the VME64AUTO power-up option.

Table A.45 : VMEbus Interrupt Status Register (VINT_STAT)

Register Name: VINT_STAT								Offset: 314

Bits	Function							
31-24	Universe Reserved							
23-16								
15-08	Reserved			SW_INT	Reserved	VERR	LERR	DMA
07-00	LINT7	LINT6	LINT5	LINT4	LINT3	LINT2	LINT1	LINT0

VINT_STAT Description

Name	Type	Reset By	Reset State	Function
SW_INT	R/Write 1 to Clear	all	Power-up Option	VME Software Interrupt Status/Clear 0=no VME Software Interrupt 1=VME Software Interrupt active
VERR	R/Write 1 to Clear	all	0	VERR Interrupt Status/Clear 0=noVME VERR Interrupt 1=VME VERR Interrupt active
LERR	R/Write 1 to Clear	all	0	LERR Interrupt Status/Clear 0=noVME LERR Interrupt 1=VME LERR Interrupt active
DMA	R/Write 1 to Clear	all	0	DMA Interrupt Status/Clear 0=noVME DMA Interrupt 1=VME DMA Interrupt active
LINT7-LINT0	R/Write 1 to Clear	all	0	LINT*x* Interrupt Status/Clear 0=no LINT*x* Interrupt 1=LINT*x* Interrupt active

SW_INT can be set with the VME64AUTO power-up option.

Table A.46 : VME Interrupt Map 0 Register (VINT_MAP0)

Register Name: VINT_MAP0				Offset: 31C

Bits	Function			
31-24	Reserved	LINT7	Reserved	LINT6
23-16	Reserved	LINT5	Reserved	LINT4
15-08	Reserved	LINT3	Reserved	LINT2
07-00	Reserved	LINT1	Reserved	LINT0

VINT_MAP0 Description

Name	Type	Reset By	Reset State	Function
LINT7-LINT0	R/W	all	0	VMEbus destination of PCI bus interrupt source

The two VME Interrupt Map registers map various interrupt sources to one of the seven VMEbus interrupt pins. A value of 001 maps the corresponding interrupt source to VIRQ*[1], a value of 002 maps to VIRQ*[2], etc. A value of 000 effectively masks the interrupt since there is no corresponding VIRQ*[0].

Table A.47 : VME Interrupt Map 1 Register (VINT_MAP1)

Register Name: VINT_MAP1				Offset: 31C

Bits	Function			
31-24	Reserved			
23-16	Reserved			SW_INT
15-08	Reserved			VERR
07-00	Reserved	LERR	Reserved	DMA

VINT_MAP1 Description

Name	Type	Reset By	Reset State	Function
SW_INT	R/W	all	Power-up Option	VMEbus Software interrupt destination
VERR	R/W	all	0	VMEbus Error interrupt destination
LERR	R/W	all	0	PCI bus Error interrupt destination
DMA	R/W	all	0	DMA interrupt destination

The two VME Interrupt Map registers map various interrupt sources to one of the seven VMEbus interrupt pins. A value of 001 maps the corresponding interrupt source to VIRQ*[1], a value of 002 maps to VIRQ*[2], etc. A value of 000 effectively masks the interrupt since there is no corresponding VIRQ*[0].

SW_INT is set to 010 with the VME64AUTO power-up option.

Table A.48 : Interrupt STATUS/ID Out (STATID)

Register Name: STATID			Offset: 328
Bits	**Function**		
31-24	STATID [7:1]	Reserved	
23-16	Universe Reserved		
15-08	Universe Reserved		
07-00	Universe Reserved		

STATID Description

Name	Type	Reset By	Reset State	Function
STATID [7:1]	R/W	all	1111111	Bits [7:1] of the STATUS/ID byte are returned when the Universe responds to a VMEbus IACK cycle.

When the Universe responds to an interrupt acknowledge cycle on VMEbus it returns an 8 bit STATUS/ID. The upper seven bits are specied by STATID eld and can be written by software to uniquely identify the VMEbus module within the system. The low order bit of the STATUS/ID is cleared if the Universe is generating a software interrupt (SW_IACK) at the same level as the interrupt acknowledge cycle, otherwise it is set.

The reset state is designed to support the VME64 Auto ID STATUS/ID value.

Table A.49 : VIRQ1 STATUS/ID Register (V1_STATID)

Register Name: V1_STATID			Offset: 324
Bits	**Function**		
31-24	Universe Reserved		
23-16	Universe Reserved		
15-08	Universe Reserved		ERR
07-00	STATID [7:0]		

V1_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 1 VMEbus interrupt

The Vx_STATID registers are read-only registers that hold the 8-bit VMEbus STATUS/ID that is acquired when the Universe performs a IACK cycle for a given interrupt level. The Universe is enabled as the interrupt handler for a given interrupt level via the VIRQx bits of the LINT_EN register. Once a vector for a given level is acquired, the Universe does not perform a subsequent interrupt acknowledge cycle at that level until the corresponding VIRQx bit in the LINT_STAT register is cleared.

The acquisition of a level *x* STATUS/ID by the Universe updates the STATUS/ID eld of the corresponding Vx_STATID register and generation of a PCI interrupt. A VMEbus error during the acquisition of the STATUS/ID vector sets the ERR bit, which means the STATUS/ID eld may not contain a valid vector.

Table A.50 : VIRQ2 STATUS/ID Register (V2_STATID)

Register Name: V2_STATID		Offset: 328

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V2_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 2 VMEbus interrupt

Table A.51 : VIRQ3 STATUS/ID Register (V3_STATID)

Register Name: V3_STATID	Offset: 32C

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V3_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 3VMEbus interrupt

Table A.52 : VIRQ4 STATUS/ID Register (V4_STATID)

Register Name: V4_STATID	Offset: 330

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V4_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 4 VMEbus interrupt

Table A.53 : VIRQ5 STATUS/ID Register (V5_STATID)

Register Name: V5_STATID		Offset: 334

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V5_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 5 VMEbus interrupt

Table A.54 : VIRQ6 STATUS/ID Register (V6_STATID)

Register Name: V6_STATID	Offset: 33B

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V6_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 6 VMEbus interrupt

Table A.55 : VIRQ7 STATUS/ID Register (V7_STATID)

Register Name: V7_STATID	Offset: 33C

Bits	Function	
31-24	Universe Reserved	
23-16	Universe Reserved	
15-08	Universe Reserved	ERR
07-00	STATID [7:0]	

V7_STATID Description

Name	Type	Reset By	Reset State	Function
ERR	R	all	0	Error Status Bit 0=STATUS/ID was acquired without bus error 1=bus error occurred during acquisition of the STATUS/ID
STATID [7:0]	R	all	0	STATUS/ID acquired during IACK cycle for level 7 VMEbus interrupt

Table A.56 : Master Control Register (MAST_CTL)

Register Name: MAST_CTL						Offset: 400

Bits	Function					
31-24	MAXRTRY			PWON		
23-16	VRL	VRM	VREL	VOWN	VOWN_ACK	Universe Reserved
15-08	Universe Reserved		PABS	Universe Reserved		
07-00	BUS_NO					

MAST_CTL Description

Name	Type	Reset By	Reset State	Function
MAXRTRY [3:0]	R/W	all	1000	Maximum Number of Retries 0000=Retry Forever, Multiples of 64 (0001 through 1111). Maximum Number of retries before the PCI master interface signals error condition
PWON [3:0]	R/W	all	0000	Posted Write Transfer Count 0000=128 bytes, 0001=256 bytes, 0010=512 bytes, 0011=1024 bytes, 0100=2048 bytes, 0101=4096 bytes, Others=Reserved Transfer count at which the PCI Slave Channel Posted Writes FIFO gives up the VME Master Interface.
VRL [1:0]	R/W	all	11	VMEbus Request Level 00=Level 0,01=Level 1,10=Level 2, 11=Level 3
VRM	R/W	all	0	VMEbus Request Mode 0=Demand,1=Fair
VREL	R/W	all	0	VMEbus Release Mode 0=Release When Done, 1=Release on Request
VOWN	W	all	0	VME Ownership Bit 0=Release VMEbus, 1=Acquire and Hold VMEbus
VOWN_ACK	R	all	0	VME Ownership Bit Acknowledge 0=VMEbus not owned, 1=VMEbus acquired and held due to assertion of VOWN
PABS	R/W	all	0	PCI Aligned Burst Size 0=32-byte, 1=64-byte Controls the PCI address boundary at which the Universe breaks up a PCI transaction. See "Posted Writes" on page 2-38. and "FIFO Operation and Bus Ownership" on page 2-82.
BUS_NO [7:0]	R/W	all	0000 0000	PCI Bus Number

VOWN_ACK is synchronized to the PCI clock.

VMEbus masters should not set the VOWN bit since this will lock up the VMEbus.

BUS_NO is used by the PCI master interface when it is mapping VME transactions into PCI Conguration space. If the bus number of the PCI address (bits [23:16]) is equal to the BUS_NO eld, then the Universe generates a Type 0 conguration cycle, otherwise Type 1 is generated.

Table A.57 : Miscellaneous Control Register (MISC_CTL)

Register Name: MISC_CTL								Offset: 404

Bits	Function							
31-24	VBTO				Reserved	VARB	VARBTO	
23-16	SW_LRST	SW_SRST	Reserved	BI	ENGBI	RESCIND	SYSCON	V64AUTO
15-08	Universe Reserved							
07-00	Universe Reserved							

MISC_CTL Description

Name	Type	Reset By	Reset State	Function
VBTO	R/W	all	0011	VME Bus Time-out 0000=Disable, 0001=16 µsec, 0010=32 µsec, 0011=64 µsec, 0100=128 µsec, 0101=256 µsec, 0110=512 µsec, 0111=1024 µsec, others=RESERVED
VARB	R/W	all	0	VMEbus Arbitration Mode 0=Round Robin, 1=Priority
VARBTO	R/W	all	01	VMEbus Arbitration Time-out 00=Disable Timer, 01=16 µs, 10=256 µs, others=Reserved
SW_LRST	W	all	0	Software PCI Reset 0=No affect, 1=Initiate LRST# A read always returns 0.
SW_SYSRST	W	all	0	Software VMEbus SYSRESET 0=No affect, 1=Initiate SYSRST* A read always returns 0.
BI	R/W	all	Power-up Option	BI-Mode 0=Universe is not in BI-Mode, 1=Universe is in BI-Mode Write to this bit to change the Universe BI-Mode status. This bit is also affected by the global BI-Mode initiator VRIRQ1*, if this feature is enabled.
ENGBI	R/W	all	0	Enable Global BI-Mode Initiator 0=Assertion of VIRQ1 ignored, 1=Assertion of VIRQ1 puts device in BI-Mode
RESCIND	R/W	all	1	Rescinding DTACK Enable 0=Disable, 1=Enable
SYSCON	R/W	all	Power-up Option	SYSCON 0=Universe is not VMEbus System Controller, 1=Universe is VMEbus System Controller
V64AUTO	R/W	all	Power-up Option	VME64 Auto ID Write: 0=No effect, 1=Initiate sequence This bit initiates Universe VME64 Auto ID Slave participation.

VMEbus masters should not write to SW_SYSRST, and PCI masters should not write to SW_LRST.

The bits VBTO, VARB and VARBTO support SYSCON functionality.

Universe participation in the VME64 Auto ID mechanism is controlled by the VME64AUTO bit. When this bit is detected high, the Universe uses the SW_IACK mechanism to generate VXIRQ2 on the VMEbus, then releases VXSYSFAIL. Access to the CR/CSR image is enabled when the level 2 interrupt acknowledge cycle completes. This sequence can be initiated with a power-up option or by software writing a 1 to this bit.

Table A.58 : Miscellaneous Status Register (MISC_STAT)

Register Name: MISC_STAT				Offset: 40h

Bits	Function							
31-24	ENDIAN	LCLSIZE	Reserved		DY4AUTO	Universe Reserved		
23-16	Universe Reserved		MYBBSY	Reserved	DY4_DONE	TXFE	RXFE	Reserved
15-08	DY4AUTOID							
07-00	Universe Reserved							

MISC_STAT Description

Name	Type	Reset By	Reset State	Function
ENDIAN	R	all	0	The Universe is always little endian on the PCI bus
LCLSIZE	R	all	Power-up Option	PCI Bus Size At the trailing edge of RST#, the Universe samples REQ64# to determine the PCI bus size. This bit reects the result. 0=32-bits, 1=64-bits
DY4AUTO	R	all	Power-up Option	DY4 Auto ID Enable 0=Disable, 1=Enable
MYBBSY	R	all	1	Universe BBSY 0=Asserted,1=Negated
DY4DONE	R	all	0	DY4 Auto ID Done 0=Not done,1=Done
TXFE	R	all	1	Transmit FIFO Empty 0=Not empty, 1=Empty
RXFE	R	all	1	Receive FIFO Empty 0=Not Empty, 1=Empty
DY4AUTOID	R	none	x	DY4 Auto ID

Table A.59 : User AM Codes Register (USER_AM)

Register Name: USER_AM	Offset: 40C

Bits	Function	
31-24	USER1AM	Universe Reserved
23-16	USER2AM	Universe Reserved
15-08	Universe Reserved	
07-00	Universe Reserved	

USER_AM Description

Name	Type	Reset By	Reset State	Function
USER1AM [5:0]	R/W	all	010000	User AM Code 1
USER2AM [5:0]	R/W	all	010000	User AM Code 2

The reset state is one of the VME64 user-dened AM codes.

Table A.60 : VMEbus Slave Image 0 Control (VSI0_CTL)

Register Name: VSI0_CTL								Offset: F00
Bits	**Function**							
31-24	EN	PWEN	PREN	Universe Reserved				
23-16	PGM		SUPER	Reserved		VAS		
15-08	Universe Reserved							
07-00	LD64EN	LLRMW	Universe Reserved					LAS

VSI0_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
PREN	R/W	all	0	Prefetch Read Enable 0=Disable, 1=Enable
PGM	R/W	all	11	Program/Data AM Code 00=Reserved, 01=Data, 10=Program, 11=Both
SUPER	R/W	all	11	Supervisor/User AM Code 00=Reserved, 01=Non-Privileged, 10=Supervisor, 11=Both
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=Reserved, 110=User1, 111=User2
LD64EN	R/W	all	0	Enable 64-bit PCI bus Transactions 0=Disable, 1=Enable
LLRMW	R/W	all	0	Enable PCI bus Lock of VMEbus RMW 0=Disable, 1=Enable
LAS	R/W	all	0	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Conguration Space, 11=Reserved

This register provides the general, VMEbus and PCI controls for this slave image. Note that only transactions destined for PCI Memory space are decoupled (the posted write RXFIFO generates on Memory space transactions on the PCI bus). This image has 4 Kbyte resolution.

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Table A.3).

Table A.61 : VMEbus Slave Image 0 Base Address Register (VSI0_BS)

Register Name: VSI0_BS	Offset: F04

Bits	Function	
31-24	BS	
23-16	BS	
15-08	BS	Universe Reserved
07-00	Universe Reserved	

VSI0_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:12]	R/W	none	x	Base Address

The base address species the lowest address in the address range that will be decoded.

Table A.62 : VMEbus Slave Image 0 Bound Address Register (VSI0_BD)

Register Name: VSI0_BD	Offset: F08

Bits	Function	
31-24	BD	
23-16	BD	
15-08	BD	Universe Reserved
07-00	Universe Reserved	

VSI0_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:12]	R/W	none	x	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register.

Table A.63 : VMEbus Slave Image 0 Translation Offset (VSI0_TO)

Register Name: VSI0_TO	Offset: F0C

Bits	Function	
31-24	TO	
23-16	TO	
15-08	TO	Universe Reserved
07-00	Universe Reserved	

VSI0_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:12]	R/W	none	x	Translation Offset

Table A.64 : VMEbus Slave Image 1 Control (VSI1_CTL)

Register Name: VSI1_CTL	Offset: F14

Bits	Function							
31-24	EN	PWEN	PREN	Universe Reserved				
23-16	PGM	SUPER	Reserved			VAS		
15-08	Universe Reserved							
07-00	LD64EN	LLRMW	Universe Reserved					LAS

VSI1_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
PREN	R/W	all	0	Prefetch Read Enable 0=Disable, 1=Enable
PGM	R/W	all	11	Program/Data AM Code 00=Reserved, 01=Data, 10=Program, 11=Both
SUPER	R/W	all	11	Supervisor/User AM Code 00=Reserved, 01=Non-Privileged, 10=Supervisor, 11=Both
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=Reserved, 110=User1, 111=User2
LD64EN	R/W	all	1	Enable 64-bit PCI bus Transactions 0=Disable, 1=Enable
LLRMW	R/W	all	1	Enable PCI bus Lock of VMEbus RMW 0=Disable, 1=Enable
LAS	R/W	all	0	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Conguration Space, 11=Reserved

This register provides the general, VMEbus and PCI controls for this slave image. Note that only PCI Memory space accesses are decoupled (the posted write RXFIFO generates PCI transactions only to Memory space). This image has a 64 Kbyte resolution (maximum size of image is 4 GBytes).

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Table A.3).

Table A.65 : VMEbus Slave Image 1 Base Address Register (VSI1_BS)

Register Name: VSI1_BS	Offset: F18

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

VSI1_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

The base address species the lowest address in the address range that will be decoded.

Table A.66 : VMEbus Slave Image 1 Bound Address Register (VSI1_BD)

Register Name: VSI1_BD	Offset: E1C

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

VSI1_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

The addresses decoded in a slave image are those which are greater than or equal to the base address and less than the bound register.

Table A.67 : VMEbus Slave Image 1 Translation Offset (VSI1_TO)

Register Name: VSI1_TO	Offset: F20

Bits	Function
31-24	TO
23-16	TO
15-08	Universe Reserved
07-00	Universe Reserved

VSI1_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:16]	R/W	none	x	Translation Offset

Table A.68 : VMEbus Slave Image 2 Control (VSI2_CTL)

Register Name: VSI2_CTL							Offset: F20

Bits	Function					
31-24	EN	PWEN	PREN	Universe Reserved		
23-16	PGM	SUPER	Reserved		VAS	
15-08	Universe Reserved					
07-00	LD64EN	LLRMW	Universe Reserved			LAS

VSI2_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
PREN	R/W	all	0	Prefetch Read Enable 0=Disable, 1=Enable
PGM	R/W	all	11	Program/Data AM Code 00=Reserved, 01=Data, 10=Program, 11=Both
SUPER	R/W	all	11	Supervisor/User AM Code 00=Reserved, 01=Non-Privileged, 10=Supervisor, 11=Both
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=Reserved, 110=User1, 111=User2
LD64EN	R/W	all	0	Enable 64-bit PCI bus Transactions 0=Disable, 1=Enable
LLRMW	R/W	all	0	Enable PCI bus Lock of VMEbus RMW 0=Disable, 1=Enable
LAS	R/W	all	0	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Conguration Space, 11=Reserved

This register provides the general, VMEbus and PCI controls for this slave image. Note that only PCI Memory space accesses are decoupled (the posted write RXFIFO generates PCI transactions only to Memory space). This image has a 64 Kbyte resolution (maximum size of image is 4 GBytes).

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Table A.3).

Table A.69 : VMEbus Slave Image 2 Base Address Register (VSI2_BS)

Register Name: VSI2_BS	Offset: F2C

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

VSI2_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.70 : VMEbus Slave Image 2 Bound Address Register (VSI2_BD)

Register Name: VSI2_BD	Offset: F30

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

VSI2_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

Table A.71 : VMEbus Slave Image 2 Translation Offset (VSI2_TO)

Register Name: VSI2_TO	Offset: F34

Bits	Function
31-24	TO
23-16	TO
15-08	Universe Reserved
07-00	Universe Reserved

VSI2_TO Description

Name	Type	Reset By	Reset State	Function
TO[31:16]	R/W	none	x	Translation Offset

Table A.72 : VMEbus Slave Image 3 Control (VSI3_CTL)

Register Name: VSI3_CTL	Offset: F3C

Bits	Function							
31-24	EN	PWEN	PREN	Universe Reserved				
23-16	PGM		SUPER	Reserved		VAS		
15-08	Universe Reserved							
07-00	LD64EN	LLRMW	Universe Reserved					LAS

VSI3_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	0	Image Enable 0=Disable, 1=Enable
PWEN	R/W	all	0	Posted Write Enable 0=Disable, 1=Enable
PREN	R/W	all	0	Prefetch Read Enable 0=Disable, 1=Enable
PGM	R/W	all	11	Program/Data AM Code 00=Reserved, 01=Data, 10=Program, 11=Both
SUPER	R/W	all	11	Supervisor/User AM Code 00=Reserved, 01=Non-Privileged, 10=Supervisor, 11=Both
VAS	R/W	all	0	VMEbus Address Space 000=A16, 001=A24, 010=A32, 011= Reserved, 100=Reserved, 101=Reserved, 110=User1, 111=User2
LD64EN	R/W	all	0	Enable 64-bit PCI bus Transactions 0=Disable, 1=Enable
LLRMW	R/W	all	0	Enable PCI bus Lock of VMEbus RMW 0=Disable, 1=Enable
LAS	R/W	all	0	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Conguration Space, 11=Reserved

This register provides the general, VMEbus and PCI controls for this slave image. Note that only PCI Memory space accesses are decoupled (the posted write RXFIFO generates PCI transactions only to Memory space). This image has a 64 Kbyte resolution (maximum size of image is 4 GBytes).

In order for a VMEbus slave image to respond to an incoming cycle, the PCI master interface must be enabled (bit BM in the PCI_CSR register, Table A.3).

Table A.73 : VMEbus Slave Image 3 Base Address Register (VSI3_BS)

Register Name: VSI3_BS	Offset: F40

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

VSI3_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.73 : VMEbus Slave Image 3 Base Address Register (VSI3_BS)

Register Name: VSI3_BS	Offset: F40

Bits	Function
31-24	BS
23-16	BS
15-08	Universe Reserved
07-00	Universe Reserved

VSI3_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:16]	R/W	none	x	Base Address

Table A.74 : VMEbus Slave Image 3 Bound Address Register (VSI3_BD)

Register Name: VSI3_BD	Offset: F44

Bits	Function
31-24	BD
23-16	BD
15-08	Universe Reserved
07-00	Universe Reserved

VSI3_BD Description

Name	Type	Reset By	Reset State	Function
BD[31:16]	R/W	none	x	Bound Address

Table A.76 : VMEbus Register Access Image Control Register (VRAI_CTL)

Register Name: VRAI_CTL	Offset: F70

Bits	Function			
31-24	EN	Universe Reserved		
23-16	PGM	SUPER	Universe Reserved	VAS
15-08	Universe Reserved			
07-00	Universe Reserved			

VRAI_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	all	Power-up Option	Image Enable 0=Disable, 1=Enable
PGM	R/W	all	11	Program/Data AM Code 00=Reserved, 01=Data, 10=Program, 11=Both
SUPER	R/W	all	11	Supervisor/User AM Code 00=Reserved, 01=Non-Privileged, 10=Supervisor, 11=Both
VAS	R/W	all	Power-up Option	VMEbus Address Space 00=A16, 01=A24, 10=A32, all others are reserved

The VME Register Access Image allows access to the Universe registers with standard VMEbus cycles. Only single cycle and lock AM codes are accepted. When a register is accessed with a RMW, it is locked for the duration of the transaction.

Table A.77 : VMEbus Register Access Image Base Address Register (VRAI_BS)

Register Name: VRAI_BS	Offset: F74

Bits	Function	
31-24	BS	
23-16	BS	
15-08	BS	Universe Reserved
07-00	Universe Reserved	

VRAI_BS Description

Name	Type	Reset By	Reset State	Function
BS[31:12]	R/W	all	Power-up Option	The base address species the lowest address in the 4 Kbyte VMEbus Register Access Image.

The base address species the lowest address in the 4 Kbyte VMEbus Register Access Image. The reset state is a function of the Power-up Option behaviour of the VAS eld in VRAI_CTL:

VRAI_CTL: VAS	BS [31:24]	BS [23:16]	BS [15:12]
A16	0	0	Power-up Option VA [28:25]
A24	0	Power-up Option VA [28:21]	0
A32	Power-up Option VA [28:21]	0	0

Table A.78 : VMEbus CSR Control Register (VCSR_CTL)

Register Name: VCSR_CTL			Offset: F80
Bits	**Function**		
31-24	EN	Universe Reserved	
23-16	Universe Reserved		
15-08	Universe Reserved		
07-00	Universe Reserved		LAS

VCSR_CTL Description

Name	Type	Reset By	Reset State	Function
EN	R/W	PWR,VME	0	Image Enable 0=Disable, 1=Enable
LAS	R/W	all	Power-up Option	PCI bus Address Space 00=PCI bus Memory Space, 01=PCI bus I/O Space, 10=PCI bus Conguration Space, 11=Reserved

The EN bit allows software to enable or disable the Universe CR/CSR image. This image can also be enabled by the VME64 Auto ID process.

For CSR's not supported in the Universe and for CR accesses, the LAS eld determines the PCI bus command that will be generated when the cycle is mapped to the PCI bus.

Table A.79 : VMEbus CSR Translation Offset (VCSR_TO)

Register Name: VCSR_TO			Offset: F84

Bits	Function	
31-24	TO	
23-16	TO	Universe Reserved
15-08	Universe Reserved	
07-00	Universe Reserved	

VCSR_TO Description

Name	Type	Reset By	Reset State	Function
TO [31:24]	R/W	PWR, VME	0	Translation Offset
TO [23:19]	R/W	all	Power-up Option	Translation Offset

For CSR's not supported in the Universe and for CR accesses, the translation offset is added to the 24-bit VMEbus address to produce a 32-bit PCI bus address. Negative offsets are not supported.

Table A.80 : VMEbus AM Code Error Log (V_AMERR)

Register Name: V_AMERR	Offset: F88

Bits	Function			
31-24	AMERR		IACK	M_ERR
23-16	V_STAT	Universe Reserved		
15-08	Universe Reserved			
07-00	Universe Reserved			

V_AMERR Description

Name	Type	Reset By	Reset State	Function
AMERR [5:0]	R	all	0	VMEbus AM Code Error Log
IACK	R	all	0	VMEbus IACK Signal
M_ERR	R	all	0	Multiple Error Occurred 0=Single error, 1=At least one error has occurred since the logs were frozen
V_STAT	R/W	all	0	VME Error Log Status Reads: 0=logs invalid, 1=logs are valid and error logging halted Writes: 0=no effect, 1=clears V_STAT and enables error logging

The Universe VMEbus Master Interface is responsible for logging the parameters of a posted write transaction that results in a bus error. This register holds the address modier code and the state of the IACK* signal. The register contents are qualied by the V_STAT bit.

Table A.81 : VMEbus Address Error Log (VAERR)

Register Name: VAERR	Offset: F8C

Bits	Function
31-24	VAERR
23-16	VAERR
15-08	VAERR
07-00	VAERR

VAERR Description

Name	Type	Reset By	Reset State	Function
VAERR [31:0]	R	all	0	VMEbus address error log

The Universe PCI Master Interface is responsible for logging the parameters of a posted write transaction that results in a bus error. This register holds the address. The register contents are qualied by the V_STAT bit of the V_AMERR register.

Table A.82 : VMEbus CSR Bit Clear Register (VCSR_CLR)

Register Name: VCSR_CLR	Offset: FF4

Bits	Function			
31-24	RESET	SYSFAIL	FAIL	Universe Reserved
23-16	Universe Reserved			
15-08	Universe Reserved			
07-00	Universe Reserved			

VCSR_CLR Description

Name	Type	Reset By	Reset State	Function
RESET	R/W	PWR, VME	0	Board Reset Reads: 0=LRST# not asserted, 1=LRST# asserted Writes: 0=no effect, 1=negate LRST#
SYSFAIL	R/W	all	Power-up Option	VMEbus SYSFAIL Reads: 0=VXSYSFAIL not asserted, 1=VXSYSFAIL asserted Writes:0=no effect, 1=negate VXSYSFAIL
FAIL	R	PWR, VME	0	Board Fail 0=Board has not failed

This register implements the Bit Clear Register as dened in the VME64 specication. Note that the RESET bit can be written to only from the VMEbus.

Table A.83 : VMEbus CSR Bit Set Register (VCSR_SET)

Register Name: VCSR_SET				Offset: FF8

Bits	Function			
31-24	RESET	SYSFAIL	FAIL	Universe Reserved
23-16	Universe Reserved			
15-08	Universe Reserved			
07-00	Universe Reserved			

VCSR_SET Description

Name	Type	Reset By	Reset State	Function
RESET	R/W	PWR, VME	0	Board Reset Reads: 0=LRST# not asserted, 1=LRST# asserted Writes: 0=no effect, 1=assert LRST#
SYSFAIL	R/W	all	Power-up Option	VMEbus SYSFAIL Reads: 0=VXSYSFAIL not asserted, 1=VXSYSFAIL asserted Writes:0=no effect, 1=assert VXSYSFAIL
FAIL	R	PWR, VME	0	Board Fail 0=Board has not failed

This register implements the Bit Set Register as dened in the VME64 specication. Note that the RESET bit can be written to only from the VMEbus. Writing 1 to the RESET bit asserts LRST#. The PCI reset remains asserted until a 1 is written to the RESET bit of the VCSR_CLR register.

Table A.84 : VMEbus CSR Base Address Register (VCSR_BS)

Register Name: VCSR_BS	Offset: FFC

Bits	Function	
31-24	BS	Universe Reserved
23-16	Universe Reserved	
15-08	Universe Reserved	
07-00	Universe Reserved	

VCSR_BS Description

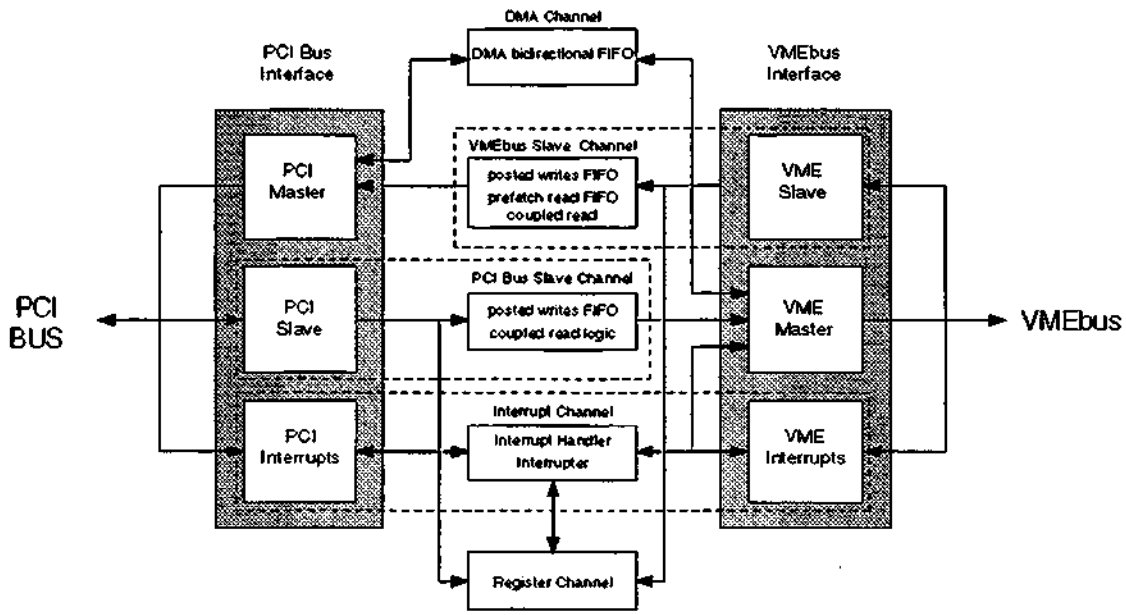
Name	Type	Reset By	Reset State	Function
BS [23:19]	R/W	PWR, VME	0	Base Address

The base address species one of thirty-one available CR/CSR windows as dened in the VME64 specication. Each window consumes 512 Kbytes of CR/CSR space.

APPENDIX B Universe Block Diagram

Appendix Scope

This appendix gives a functional block diagram of the Universe VME Interface Controller. The relationships between PCI Master and PCI Slave Modules, VMEbus Master and VMEbus Slave Modules, and VME/PCI Slave Channels are shown. Also, the Interrupt Channel and Register Channel are illustrated.



APPENDIX C VME BIOS Routines

Appendix Scope

This appendix describes in detail, on a function by function basis, the routines which comprise the V5A VME BIOS. The following VME BIOS functions are accessible from software interrupt 69H:

Note: all function numbers are given in hexadecimal

```
************************************************************************
function 00h - GET_VME_BIOS_VERSION
Returns the VME BIOS version, board type and A32 capability

ENTRY: AH = 00H
EXIT:  AL = VERSION OF THIS FIRMWARE, VERSION & MAJOR REV, IN BCD format
       AH = VME SYSTEM/OPTION INFO.:

              bit:        if set:                 if clear:
              7-5         Reserved
              4-1         0000 = XT
                          0001 = XP
                          0010 = XPS
                          0011 = XPX
                          0100 = VME-1486
                          0101 = VME-7486
                          0110 = VME-2486
                          0111 = VME-8486
                          1000 = VME-3486
                          1001 = VME-9586
                          1010 = V5A
                          1011 = VME-190
                          1100 = IV-586
              0           EXTENDED ADDRESS        NO EXTENDED
                          CAPABLE                 ADDRESS CAPABILITY
```

```
*************************************************************************
function 01h - REQUEST_VME_INTERRUPT
Function generates a VME IRQ to the VMEbus.

ENTRY: AH = 01H
       AL = VMEBUS INTERRUPT REQUEST LEVEL (1-7)
       BL = VMEBUS INTERRUPT VECTOR NUMBER (ID/Status Byte)
EXIT:  NOTHING


*************************************************************************
function 02h - GET_VME_INTERRUPT_STATUS
Function returns the pending VME IRQ interrupt status.

ENTRY: AH = 02H
EXIT:  AL = VMEBUS INTERRUPT LEVEL PENDING, 0 returned if none pending,

*************************************************************************
function 03h - SETUP_VME_INTERRUPTS
Function enables VME IRQs to pass through to the PCI (local) bus.

ENTRY: AH = 03H
       AL = any bit 1-7 set enables the corresponding VME IRQ
       BL = for each bit set in AL, the corresponding bit in BL tells
            which PCI (local) interrupt path to take.  0 means LINT0 (PCI
            INTA), 1 means LINT1 (PCI Steerable interrupt 0).
EXIT:  NOTHING

*************************************************************************
function 0Bh - CLEAR VME INTERRUPTS
Function disables all VME interrupts

ENTRY: AH = 0BH
EXIT:  NOTHING

*************************************************************************
function 0Ch - GET_SYSFAIL_STATUS
Function returns the status of the VMEbus SYSFAIL signal

ENTRY: AH = 0CH
EXIT:  AL BIT 0 = 1 IF SYSFAIL IS ASSERTED
       AL BIT 0 = 0 IF SYSFAIL IS DEASSERTED

*************************************************************************
function 0Dh - SET_SYSFAIL_STATUS
Function will set or clear the SYSFAIL signal

ENTRY: AH = 0DH
       AL BIT 0 = 1 TO ASSERT SYSFAIL
       AL BIT 0 = 0 TO DEASSERT SYSFAIL
EXIT:  NOTHING
```

```
*************************************************************************
function 10h - SERVICE_INTERRUPTS
Function processes the pending VMEbus interrupt.

ENTRY: AH = 10H
EXIT:  AL:
           Bit  Meaning if set
            0   highest priority vme interrupt was serviced
            1   SYSFAIL Interrupt occurred
            2   ACFAIL Interrupt occurred
            3   DMA error occurred
            4   PCI error occurred
            5   VME error occurred
            6   VOWN Interrupt occurred
            7   Reserved
       BL:
          bits 0-6 = VME IRQ (1-7) acknowleged
           bit 7    = BUS error occurred after int vector was read
       CL: VME Interrupt Status/ID byte;  0 if bit-5 in AL is set
       EOI's acknowleged.

*************************************************************************
function 12h - GET_MASTER_IMAGE
Function will return information about a VME Master Image.

ENTRY: AH = 12H
       AL = image number (0-3)

EXIT:  AL = control status:
             bit 7 = image enabled
             bit 6 = Posted Write enabled
             bits 5-3 = VME address space (000 = A16, 001 = A24,
                              010 = A32, 110 = user1, 111 = user2)
             bit 2 = Data/PGM    (0 = data, 1 = program)
             bit 1  = Privilege (0 = user, 1 = supervisory)
             bit 0  = Block Transfer enabled
       BX = A31-A16 of Base address
       CX = A31-A16 of Bound address
       DX = A31-A16 of translation offset
       AH = A15-A12 of Base address in bits 7-4, A15-A12 of Bound
                    address in bits 3-0 (image 0 only)
       DI = A15-A12 of Translation offset in bits 3-0 (image 0 only)

*************************************************************************
function 13h - PUT_MASTER_IMAGE
Function will setup parameters for a VME Master Image

ENTRY: AH = 13H
       AL = image number (0-3)

       DI = control status:
              bit 7 = enable image
              bit 6 = Posted Write enabled
              bits 5-3 = VME address space (000 = A16, 001 = A24,
                               010 = A32, 110 = user1, 111 = user2)
              bit 2 = Data/PGM    (0 = data, 1 = program)
              bit 1  = Privilege (0 = user, 1 = supervisory)
              bit 0  = Block Transfer enabled
       SI (image 0 only) = odd bytes of addresses
              A15-A12 of Base address in bits 11-8, A15-A12 of Boundary
              address in bits 7-4, A15-A12 of Translation offset in bits 3-0
       BX = A31-A16 of Base address
       CX = A31-A16 of Bound address
       DX = A31-A16 of translation offset
EXIT:  NOTHING
```

```
*************************************************************************
function 14h - GET_SLAVE_IMAGE
Function will return information about a VME Slave Image

ENTRY: AH = 14H
       AL = Image number (0-3)
EXIT:  DI = control status:
            bit 10 = image enabled
            bit  9 = Posted Write enabled
            bit  8 = Prefetch Read enabled
            bits 7-5 = VME address space (000 = A16, 001 = A24,
                       010 = A32, 110 = user1, 111 = user2)
            bits 4&3 = Data/PGM (01 = data, 10 = program, 11 = both)
            bits 2&1 = Privilege (01 = user, 10 = supervisory, 11 = both)
            bit 0  = PCI Lock on VME RMW enabled
       BX = A31-A16 of Base address
       CX = A31-A16 of Bound address
       DX = A31-A16 of translation offset
       AH = A15-A12 of Base address in bits 7-4, A15-A12 of Bound
                    address in bits 3-0 (image 0 only)
       AL = A15-A12 of Translation offset in bits 3-0 (image 0 only)


*************************************************************************
function 15h - PUT_SLAVE_IMAGE
Function will setup parameters for a VME Slave Image

ENTRY: AH = 15H
       AL = Image number (0-3)
       DI = control status:
            bit 10 = enable image
            bit  9 = Posted Write enabled
            bit  8 = Prefetch Read enabled
            bits 7-5 = VME address space (000 = A16, 001 = A24,
                       010 = A32, 110 = user1, 111 = user2)
            bits 4&3 = Data/PGM (01 = data, 10 = program, 11 = both)
            bits 2&1 = Privilege (01 = user, 10 = supervisory, 11 = both)
            bit 0  = Enable PCI Lock on VME RMW
       BX = A31-A16 of Base address
       CX = A31-A16 of Bound address
       DX = A31-A16 of translation offset
       SI = A15-A12 of Base address in bits 7-4, A15-A12 of Bound
                    address in bits 3-0, A15-A12 of Translation
                    offset in bits 11-8 (image 0 only)
EXIT:  NOTHING

*************************************************************************
function 16h - GET_EEPROM_WORD
Function returns a word from the on-board serial EEPROM.

ENTRY: AH = 16H
       AL = Word offset into the serial EEPROM (0-63)
EXIT:  AX = EEPROM data from the requested OFFSET

*************************************************************************
function 17h - SET_EEPROM_WORD
Function writes a word to the on-board serial EEPROM.
WARNING: This device is used internally to store board information.
         Writing to the wrong location will interfere with proper board
         performance!

ENTRY: AH = 17H
       AL = Word offset into the serial EEPROM (0-63)
       BX = Data to write to the device
EXIT:  NOTHING
```

```
*************************************************************************
function 18h - GET_SERIAL_SWITCH
Function returns the 8 on-board serial switch values.

ENTRY: AH = 18H
EXIT:  AL = Bits 0-7 correspond to serial switches 1-8 respectively.

*************************************************************************
function 19h - SET_SERIAL_SWITCH
Function writes to the on-board serial switches.
WARNING: These switches are used to run certain on-board resources.
         Writing the wrong value to these switches will render the board
         inoperable.

ENTRY: AH = 19H
       AL = Bits 0-7 correspond to switches 1-8 respectively
EXIT:  NOTHING

*************************************************************************
function 1fh - SET_BOARD
Function loads all VME parameters from saved (EEPROM) values.

ENTRY: AH = 1FH
EXIT:  NOTHING

*************************************************************************
function 20h - Read VMEbus.
Read the VMEbus in real mode (up to 64Kb)

ENTRY: AH = 20H
       AL = transfer parameters:
            bits 6-5 = vdw: 00=D8  01=D16  10=D32  11=not supported
             bit 4 = mode: 0=non-privileged  1=supervisor
             bit 3 = pgm/data: 0=data  1=program
             bits 2-0 = VME address space: 000=A16  001=A24  010=A32
                                           101=CR/CSR   110=User1  111=User2
       BX = VMEbus address bits A31-A16
       SI = VMEbus address bits A15-A0
       CX = number of bytes to transfer
       DS:DI = PCI bus (local) address

EXIT:     Nothing.

*************************************************************************
function 21h - Write VMEbus.
Write to the VMEbus in real mode (up to 64Kb)

ENTRY: AH = 21H
       AL = transfer parameters:
            bits 6-5 = vdw: 00=D8  01=D16  10=D32  11=not supported
             bit 4 = mode: 0=non-privileged  1=supervisor
             bit 3 = pgm/data: 0=data  1=program
             bits 2-0 = VME address space: 000=A16  001=A24  010=A32
                                           101=CR/CSR   110=User1  111=User2
       BX = VMEbus address bits A31-A16
       DI = VMEbus address bits A15-A0
       CX = number of bytes to transfer
       DS:SI = PCI bus (local) address
```

```
EXIT:     Nothing.

*************************************************************************
function 24h - Enable Universe Registers to the VMEbus.
Function will enable the UNIVERSE registers to appear on the VMEbus

ENTRY: AH = 24H
       AL = bit 7 = enable if set, disable if clear
            bits 6&5 = pgm/data: 01=data, 10=program, 11=both
            bits 4&3 = mode: 01=non-privileged  10=privileged, 11=both
            bits 2-0 = VME address space: 000=A16  001=A24  010=A32
       BX = VMEbus address bits A31-A16
       CL = VMEbus address bits A15-A12 in CL bits 7-4

EXIT:     Nothing.
```

APPENDIX D V5A Register Summary

Appendix Scope

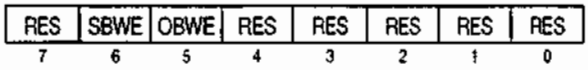
This appendix gives a summary of the V5A board-specific registers which are not standard PC/AT, or peripheral device registers. The register layout and location are given, as are the bit mneumonics and names. Read or write capability for each bit is also shown.

RES	SBWE	OBWE	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0

Reset Control Register
I/O 2A0H

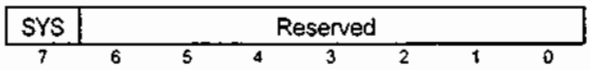
Bit	Description
0	Ignore SYSRESET from VME (R/W)
4	Front Panel Reset to VME SYSRESET Enable (R/W)

BIOS Control Register
I/O 2A1H



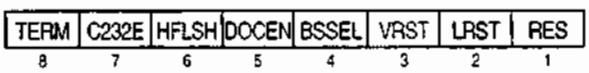
Bit	Description
5	Socketed BIOS Write Enable (R/W)
6	Onboard BIOS Write Enable (R/W)

User Register
I/O 2A2H



Bit	Description
7	SYS LED Control (R/W)

Serial EEPROM/Switch Register
I/O 2A3H



Bit	Description
0	Serial Clock (R/W)
1	Serial Data Out (R/W)
2	Serial Switch Program Enable (R/W)
3	Serial Swtich Read Enable (R/W)
4	Serial Switch Data In (R/W)
5	EEPROM Data In (R/W)
6	EEPROM Chip Select (R/W)

APPENDIX E VME Address Modifier Codes

The following table shows the standard VME address modifier codes. Codes 10H through 1FH are user-defined. All other codes are reserved.

Address Modifier	Address Size	Transfer Type
3FH	24	Standard Supervisory Block Transfer (BLT)
3EH	24	Standard Supervisory Program Access
3DH	24	Standard Supervisory Data Access
3BH	24	Standard Non-Privileged Block Transfer (BLT)
3AH	24	Standard Non-Privileged Program Access
39H	24	Standard Non-Privileged Data Access
38H	24	Standard Non-Privileged 64-bit Block Transfer (MBLT)
37H	40	A40 BLT
35H	40	A40 Lock Command (LCK)
34H	40	A40 Access
32H	24	A24 Lock Command (LCK)
2FH	N/A	Configuration ROM/Control & Status Register
2DH	16	Short Supervisory Access
2CH	16	A24 Lock Command (LCK)
29H	16	Short Non-Privileged Access
0FH	32	Extended Supervisory Block Transfer (BLT)
0EH	32	Extended Supervisory Program Access
0DH	32	Extended Supervisory Data Access
0CH	32	Extended Supervisory 64-bit Block Transfer (MBLT)
0BH	32	Extended Non-Privileged Block Transfer (BLT)
0AH	32	Extended Non-Privileged Program Access
09H	32	Extended Non-Privileged Data Access
08H	32	Extended Non-Privileged 64-bit Block Transfer (MBLT)
05H	32	A32 Lock Command (LCK)
04H	64	A64 Lock Command (LCK)
03H	64	A64 Block Transfer (BLT)
01H	64	A64 Single Transfer Access
00H	64	A64 64-bit Block Transfer (MBLT)

Table E-1 Address Modifier Codes

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP