VMIVME-7589-xx1 and -xx2

Tundra Universe™ Based VMEbus Interface Option

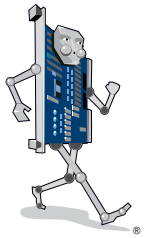
Product Manual

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(205) 880-0444 ♦ (800) 322-3616 ♦ Fax: (205) 882-0859

500-007589-001 Rev. A
17-Feb-98

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(205) 880-0444 ♦ (800) 322-3616 ♦ Fax: (205) 882-0859

COPYRIGHT AND TRADEMARKS



(I/O man figure)

REAL-TIME NETWORKING RTnet™

(IOWorks man figure)

UIOC®

WinUIOC®

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(205) 880-0444 ♦ (800) 322-3616 ♦ Fax: (205) 882-0859

Table of Contents

List of Figures

List of Tables

CHAPTER 1

General Information

Contents

1.1 Introduction

The VMIVME-7589-xx1 and -xx2 PCI-to-VMEbus interface options are proprietary PCI-to-VMEbus mezzanine boards built around the Tundra Universe interface chip (CA91C042). This manual supports both options of the product; therefore, the product is referred to as the “VMIVME-7589 VMEbus interface option” throughout the manual. The VMIVME-7589 provides the following features:

- 33 MHz PCI local bus interface with five separate PCI-to-VMEbus slave images (windows)
- High-performance 64-bit (multiplexed) VMEbus interface with four separate VMEbus-to-PCI slave images (windows)
- Programmable DMA controller with linked list support
- VMEbus system controller functionality
- Automatic slot 1 controller detection
- PCI and VMEbus interrupt generation
- VMEbus interrupt handler
- Communication support via four 1-bit mailbox registers with PCI interrupt capability

- Master/slave endian conversion hardware
- Non-slot 1 bus timeout timer

Since the interface board is built around the Tundra Universe chip, the majority of the documentation for the VMIVME-7589 VMEbus interface option is extracted from the Universe manual, itself. However, there are features of the Universe chip that are not supported at the board level, such as a 64-bit PCI, interface, JTAG testability, and certain power-up options. These unsupported features are appropriately noted within the documentation.

In addition, auxiliary functionality has been added to the VMIVME-7589 VMEbus interface option to provide various functionality such as master/slave endian conversion, non-slot 1 bus timeout timer, mail-box registers with PCI interrupt capability, and Universe errata fixes.

Programming the VMIVME-7589 VMEbus Interface Option

The VMIVME-7589 VMEbus interface option is configured by programming the auxiliary function global interrupt mask register, the external system registers, the mail-box communication registers, and the Universe Control and Status Registers. In addition, the board has four jumpers which configure the board for generating/receiving VMEbus reset, asserting SYSFAIL on power-up, and mapping the Universe registers in memory or I/O space. Refer to Chapter 2, Functional Description.

The base addresses of the various registers are located in configuration space and can be accessed using PCI BIOS calls or VMIC control function library software. Refer to Appendix A for a complete description of interface registers.

1.2 Document Overview

This document serves as both a programmer's instruction manual and a reference guide to the VMIVME-7589 VMEbus interface option. The manual is organized as follows:

- Chapter 1 - General Information - Introduces the interface option, the Universe chip, and provides the reader with information about references, concepts, and conventions required to use the manual.
- Chapter 2 - Functional Description - Describes the VMIVME-7589 VMEbus board and the Universe functions, beginning with overall functionality.
- Chapter 3 - Auxiliary Functions - Describes the programmable bus timeout timer capability of the VMIVME-7589 VMEbus interface option and the BERR* signal masking capability of the interface option.
- Chapter 4 - Endian Conversion - Includes the instructions for endian conversion.

- Chapter 5 - Errata - Describes errata associated with the Universe that must be considered when programming the VMIVME-7589 VMEbus interface option.
- Chapter 6 - PCI/VMEbus Deadlock - Describes a deadlock scenario that could occur when using the interface option and describes how to avoid this.
- Appendix A - Registers - Includes references necessary for the implementation of the VMIVME-7589 VMEbus interface option. Appendix A provides reference I/O maps and register-level descriptions of the interface option.

1.3 Tundra Corporation Reprinted Information

The information about the Tundra Universe product is provided by Tundra Corporation and is reprinted here with permission. This material is used extensively throughout this document. The only changes made to this text were made for section numbering purposes; however, VMIC-specific information injected directly into the Tundra text is preceded by the VMIC logo:

In Chapter 1, material from Tundra includes Sections 1.5 and 1.6.

All of Chapter 2 is provided by Tundra except for the following VMIC generated sections: 2.1, 2.2, 2.8.3, 2.8.4, 2.11, and 2.12.2.1.

Chapters 3, 4, 5, and 6 are exclusively VMIC.

Appendix A is a combined Tundra/VMIC chapter. VMIC information is included in:

- the auxiliary function global interrupt mask register,
- the system register section, and
- and the mailbox register section.

The Universe Control and Status Register material is a direct reprint of the Tundra-provided information.

1.4 References

For the most up-to-date physical description and specifications for the VMIVME-7589, please refer to:

VMIC specification number 800-007589-000

The following books refer to the Tundra Universe-based bridge option available in the VMIVME-7589:

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

PCI Bus Interface and Clock Distribution Chips
PLX Technology, Inc.
625 Clyde Avenue
Mountain View, CA 94043
(415) 694-2800
(415) 960-0479

There are many books widely available on the subject of general PC / AT use and programming. Some reference sources which may be particularly helpful in using or programming the VMIVME-7589 include:

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

Programmer's Guide to the AMIBIOS
American Megatrends, Inc.
6145-F Northbelt Parkway
Norcross, GA 30071

The VMIC VMEbus interrupt and control function software library references include for Windows NT:

VMISFT-9420 IOWorks Access™ User's Guide
Doc. No. 520-009420-910

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

VMEbus Specification Rev. C.1
and The VMEbus Handbook
VITA
VMEbus International Trade Association
7825 East Gelding Drive
No. 104
Scottsdale, AZ 85260
(602) 951-8866
FAX: (602) 951-0720
Internet: http://www.vita.com

1.5 Introduction to the Universe Chip

The Tundra Universe VMEbus interface chip (CA91C042) provides a reliable high-performance 64-bit VMEbus to PCI interface in one device. The Universe device results from our ongoing commitment to Openbus interface design, which has produced the SCV64 and AVICS VMEbus interfaces, and the LIFE Futurebus+ interface.

Designed in consultation with Motorola, the Universe is compliant with the VME64 specification and is tuned to the new generation of high-speed processors.

The Universe is ideally suited for CPU boards acting as both master and slave in the VMEbus system, and is particularly fitted for PCI local systems. The Universe is manufactured in a CMOS process.

1.5.1 The Features

- Fully compliant, 64-bit, 33 MHz PCI local bus interface,
- fully compliant, high-performance 64-bit VMEbus interface,
- integral FIFOs for write posting to maximize bandwidth utilization,
- programmable DMA controller with linked list support,
- complete suite of VMEbus address and data transfer modes,
  - A32/A24/A16 master and slave
  - D64 (MBLT)/D32/D16/D08 master and slave
  -BLT, ADOH, RMW, LOCK
- flexible register set, programmable from the PCI bus and the VMEbus ports,
- full VMEbus system controller functionality.

Note: The VMIVME-7589 VMEbus interface option supports only a 32-bit PCI bus.

1.5.2 Industry Proven Performance

The Universe builds on Tundra's experience with its highest performing, commercially available single chip VME64 interface, the SCV64. This performance excellence is the product of the decoupling architecture unique to Tundra's family of Bus-Bridging Components. The Universe uses built-in FIFOs to decouple the transfer of data between the VMEbus and PCI bus, thus compensating for mismatches in relative bus performance. In addition to the advantage of decoupling, the Universe offers block transfer capability (including BLT and MBLT), and an integral DMA controller.

1.5.3 Industry Proven VMEbus Attributes

Like the SCV64, the Universe implements all of the addressing and data transfer modes documented in the VME64 specification (except A64 and those intended to support 3U applications, that is, A40 and MD32). In addition, it provides full VMEbus system controller functionality with multilevel arbitration modes. All VMEbus features are fully programmable from the PCI and VMEbus side, making the Universe a flexible interface solution. With its other VMEbus functions, the Universe provides full VMEbus interrupter and interrupt handler capability, automatic system controller logic, and multilevel VMEbus requester. The Universe also supports read-modify-write operations on the VMEbus.

1.5.4 PCI: The *de facto* Local Bus Interface Standard

As a result of the computing industry's widespread acceptance of PCI, the leading semiconductor vendors have built PCI support into their newest processor and peripheral families. To allow VMEbus single-board computer vendors to capitalize on these new PCI components, Tundra has developed a PCI-to-VMEbus interface controller, the Universe. The availability of the Universe, with PCI support, eases the development of multi-master, multi-processor architectures on VMEbus systems.

1.6 Conventions

1.6.1 Signals

Signals on the VMEbus and PCI bus are either active high or active low. Active low signals are defined as true (asserted) when they are at logic low. Similarly, active high signals are defined as true at a logic high. Signals are considered asserted when active and negated when inactive, irrespective of voltage levels.

For voltage levels, the use of ‘0’ indicates a low voltage while a ‘1’ indicates a high voltage.

Table 1-1 Signal Conventions Used in this Manual

SIGNAL#	active low signals

1.6.2 Symbols

Caution: This symbol alerts the reader to procedures or operating levels which may result in misuse of or damage to the Universe.

Note: This symbol directs the reader’s attention to useful information or suggestions.

Note: This symbol indicates text built by VMIC and injected into Tundra provided information.

1.6.3 Terminology

When discussing bus ownership, this manual uses “master” to indicate the bus owner, and “slave” to indicate the address accessed by the master. These terms are used for both PCI and VMEbus descriptions. For PCI bus discussions, the reader should be aware that “master” and “slave” correspond to “initiator” and “target” in PCI terminology.

The term “cycle” refers to a single data beat, while a “transaction” is composed of one or more “cycles”.

CHAPTER
2

Functional Description

Contents

2.1 Architectural Overview of the VMIVME-7589 Interface Board

The VMIVME-7589 interface board consists of a Universe chip, a PLX9060ES PCI target interface, endian conversion muxing hardware, and miscellaneous auxiliary function hardware. These components are graphically illustrated in Figure 2-1. The majority of the interface functionality including the basic PCI-to-VMEbus interface, DMA controller functionality, and VMEbus system controller functionality is provided by the Universe chip. The PLX9060ES serves as a PCI target interface into the auxiliary functions hardware. This hardware provides the system registers, the mailbox registers, the endian conversion control logic, the non-slot 1 bus timeout timer, as well as various other miscellaneous logic.

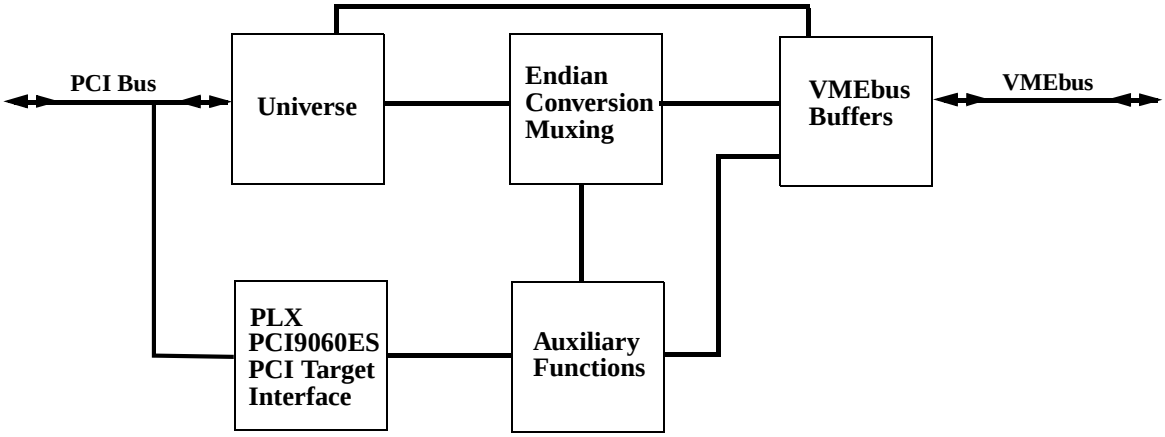


Figure 2-1 Universe-Based PCI-to-VMEbus Interface

2.2 PCI-to-VMEbus Interface Option Board Jumpers

The VMEbus interface option is tested for system operation and shipped with factory-installed header jumpers. The interface option is shipped with the VME board as part of the VMIVME-7589. Figure 2-2 illustrates the physical location of the user-configurable jumpers and connectors on the interface option board. Table 2-1 lists each jumper designator, its function, and the factory-installed default configuration.

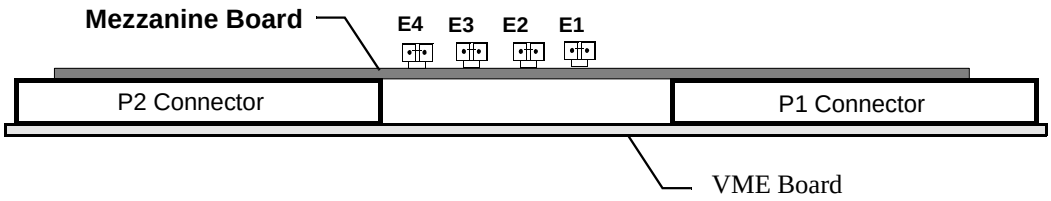


Figure 2-2 Edge view of the Jumper Locations on the Mezzanine Board

Table 2-1 VMIVME-7589 Interface Option Jumper Functions and Factory Settings

Jumper	Function	Factory Setting
E1	Installed - Universe memory mapped Removed - Universe I/O mapped	Installed (Should not be removed)
E2	Installed - Receives VMEbus SYSRESET Removed - Does not receive	Installed
E3	Installed - Drives VMEbus SYSRESET Removed - Does not drive	Installed
E4	Installed - SYSFAIL Not asserted upon reset Removed - SYSFAIL Asserted upon reset	Installed

Any other jumper locations are reserved for VMIC use only and should not be altered from the factory default settings.

Due to a Universe chip errata which maps the 4 K register set into 64 K space, the registers should be configured to be mapped into memory space. The address of the registers is contained in the Universe's PCI configuration space base address register 0 (i.e. config space offset 0x10).

2.3 The Universe Architectural Overview

This section introduces the general architecture of the Universe. This description makes frequent reference to the functional block diagram provided in Figure 2-3 on page 2-5. Notice that for each of the interfaces, VMEbus and PCI bus, there are three functionally distinct modules: master module, slave module, and interrupt module. These modules are connected to the different functional channels operating in the Universe. These channels are:

- the VME Slave Channel,
- the PCI Bus Slave Channel,
- the DMA Channel,
- the Interrupt Channel, and
- the Register Channel.

The Architectural Overview is organized into the following sections:

- “VMEbus Interface”,
- “PCI Bus Interface”,
- “Interrupter and Interrupt Handler”, and
- “DMA Controller”.

These sections describe the operation of the Universe in terms of the different modules and channels illustrated in Figure 2-3.

2.3.1 VMEbus Interface

2.3.1.1 Universe as 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 2-44). With posted write transactions, data is written to a Posted Write Receive FIFO (RXFIFO), and the VMEbus master receives data acknowledgment 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 2-17 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.

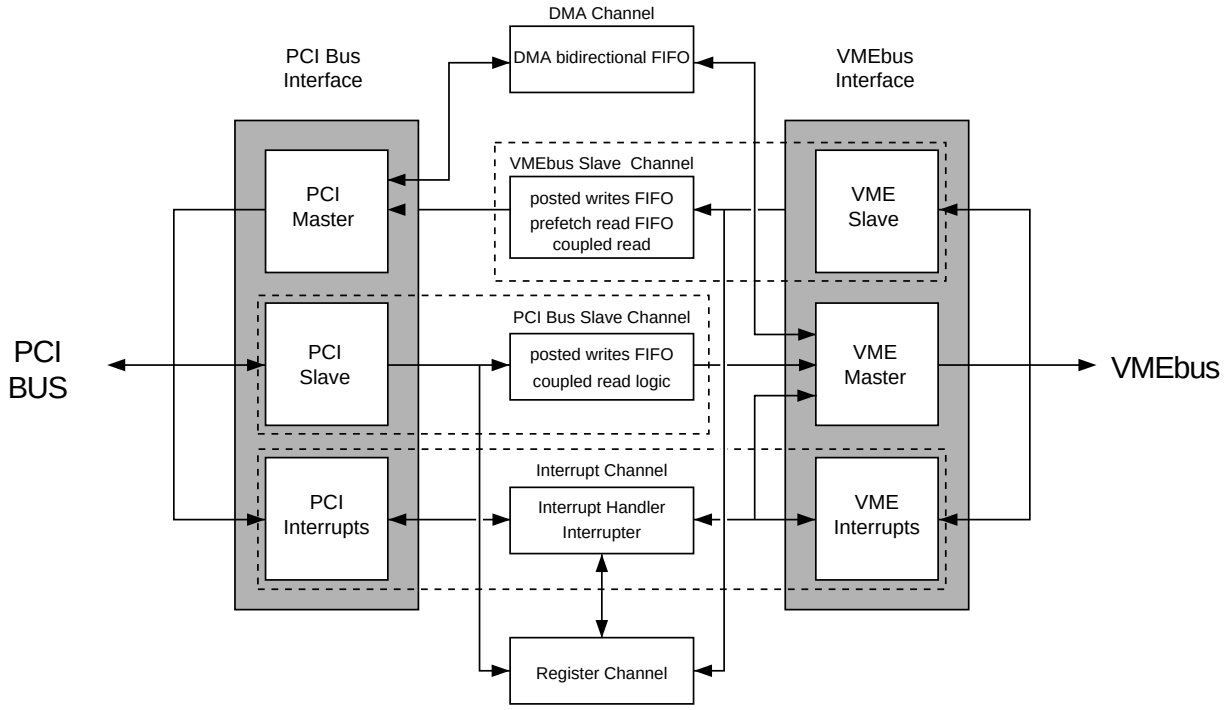


Figure 2-3 Architectural Diagram for the Universe

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.

2.3.1.2 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.

2.3.2 PCI Bus Interface

2.3.2.1 Universe as PCI Slave

Read transactions from the PCI bus are always processed as coupled. Write transactions may be either coupled or posted, depending upon the setting of the PCI bus slave image (see "PCI Bus Slave Images" on page 2-47). With a posted write transaction, write data is written to a Posted Write Transmit FIFO (TXFIFO) and the PCI bus master receives data acknowledgment from the Universe with zero wait states. Meanwhile, the Universe obtains the VMEbus and writes the data to the VMEbus resource independent of the initiating PCI master (see "Posted Writes" on page 2-38 for a full description of this operation).

To allow PCI masters to perform RMW and ADOH cycles, the Universe provides a Special Cycle Generator. The Special Cycle Generator can be used in combination with a VMEbus ownership function to guarantee PCI masters exclusive access to VMEbus resources over several VMEbus transactions (see "Exclusive Accesses" on page 2-41 and "RMW and ADOH Cycles" on page 2-40 for a full description of this functionality).

2.3.2.2 Universe as PCI Master

The Universe becomes PCI master when the PCI Master Interface is internally requested by the VME Slave Channel or the DMA Channel. There are mechanisms provided which allow the user to configure the relative priority of the VME Slave Channel and the DMA Channel.

2.3.3 Interrupter and Interrupt Handler

2.3.3.1 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 “Interrupter” on page 2-58 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 2-63).

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 “Universe PCI Interrupt Generation” on page 2-59).

2.3.3.2 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.

2.3.4 DMA Controller

The Universe provides an internal DMA controller for high performance data transfer between the PCI and VMEbus. DMA operations between the source and destination bus are decoupled through the use of a single bidirectional FIFO (DMAFIFO). Parameters for the DMA transfer are software configurable in the Universe registers (see “DMA Controller” on page 2-71).

The principal mechanism for DMA transfers is the same for operations in either direction (PCI to VME, or VME to PCI), only the relative identity of the source and destination bus changes. In a DMA transfer, the Universe gains control of the source bus and reads data into its DMAFIFO. Following specific rules of DMAFIFO operation (see “FIFO Operation and Bus Ownership” on page 2-86), it then acquires the destination bus and writes data from its DMAFIFO.

The DMA controller can be programmed to perform multiple blocks of transfers using entries in a linked-list. The DMA will work through the transfers in the linked-list following pointers at the end of each linked-list entry. Linked-list operation is initiated through a pointer in an internal Universe register, but the linked-list itself resides in PCI bus memory.

2.4 VMEbus Interface

The VMEbus interface incorporates all operations associated with the VMEbus. This includes master and slave functions, VMEbus configuration and system controller functions. These operations are covered as follows:

- "VMEbus Requester" below,
- "Universe as VME Master" on page 2-11,
- "Universe as VME Slave" on page 2-15,
- "VMEbus Configuration - First Slot Detector" on page 2-22, and
- "System Controller Functions" on page 2-22.

For information concerning the Universe VMEbus slave images, see "VME Slave Images" on page 2-44.

2.4.1 VMEbus Requester

2.4.1.1 Internal Arbitration for VMEbus Requests

Three different internal channels within the Universe require use of the VMEbus: the Interrupt Channel, the PCI Slave Channel, and the DMA Channel. These three channels do not directly request the VMEbus, instead they compete internally for ownership of the VME Master Interface.

The Interrupt Channel (refer to Figure 2-3 on page 2-5) always has the highest priority for access to the VME Master Interface. The DMA and PCI Slave Channel requests are handled in a round robin fashion. The channel awarded VMEbus mastership maintains ownership of the VMEbus until it is 'done'. The definition of 'done' for each channel is given below in "VMEbus Release" on page 2-10.

The Interrupt Channel requests the VMEbus master when it detects an enabled VMEbus interrupt line asserted and needs to run an interrupt acknowledge cycle to acquire the STATUS/ID.

The PCI Slave Channel requests the VME Master Interface to service the following conditions:

- the TXFIFO contains a complete transaction, or
- if there is a coupled cycle request.

The DMA Channel requests the VME Master Interface if:

- the DMAFIFO level reaches the programmed PCI aligned burst size (as set with PABS in the MAST_CTL register), or
- the DMA block is complete (see "DMA Controller" on page 2-71).

In the case of the DMA Channel, the user can optionally use the DMA Channel VMEbus-off-timer to further qualify requests from this channel. The VMEbus-off-timer controls how long the DMA remains off the VMEbus before making another request (see “PCI to VME Transfers” on page 2-87).

The Universe provides a software mechanism for VMEbus acquisition through the VMEbus ownership bit (VOWN in the MAST_CTL register, Table A-59). When the VMEbus ownership bit is set, the Universe acquires the VMEbus and sets an acknowledgment bit (VOWN_ACK in the MAST_CTL register, Table A-59) and optionally generates an interrupt to the PCI bus (see “Exclusive Accesses” on page 2-41). The Universe maintains VMEbus ownership until the ownership bit is cleared. During the VMEbus tenure initiated by setting the ownership bit, only the PCI Slave Channel and Interrupt Channel can access the VME Master Interface.

2.4.1.2 Request Modes

Request Levels

The Universe is software configurable to request on all VMEbus request levels: BR3*, BR2*, BR1*, and BR0*. The default setting is for level 3 VMEbus request. The request level is a global programming option set through the VRL bits in the MAST_CTL register (Table A-59). The programmed request level is used by the VME Master Interface regardless of the channel (Interrupt Channel, DMA Channel, or PCI Slave Channel) currently accessing the VME Master Interface.

Fair and Demand

The Universe requester may be programmed for either Fair or Demand mode. The request mode is a global programming option set through the VRM bits in the MAST_CTL register (Table A-59).

In Fair mode, the Universe does not request the VMEbus until there are no other VMEbus requests pending at its programmed level. This mode ensures that every requester on a given level has access to the bus.

In Demand mode (the default setting), the requester asserts its bus request regardless of the state of the BRn* line. By requesting the bus frequently, requesters far down the daisy chain may be prevented from ever obtaining bus ownership. This is referred to as “starving” those requesters. Note that in order to achieve fairness, all bus requesters in a VME system must be set to Fair mode.

2.4.1.3 VMEbus Release

The Universe VMEbus requester can be configured as either RWD (Release When Done) or ROR (Release On Request) using the VREL bit in the MAST_CTL register (Table A-59). The default setting is for RWD. ROR means the Universe releases BBSY* only if a bus request is pending from another VMEbus master and once the channel that is current owner of the VME Master Interface is done. Ownership of the bus may

be assumed by another channel without re-arbitration on the bus if there are no pending requests on any level on the VMEbus.When set for RWD, the VME Master Interface releases BBSY* when the channel accessing the VME Master Interface is done (see below). Note that the MYBBSY status bit in the MISC_STAT register (Table A-61) is set while the Universe asserts the BBSY* output.

The VMEbus is released (in RWD mode) when the channel (for example, the DMA Channel) is done, even if another channel has a request pending (for example, the PCI Slave Channel). A re-arbitration of the VMEbus is required for any pending channel requests. Each channel has a set of rules that determine when it is 'done' with its VMEbus transaction.

The Interrupt Channel is done when a single interrupt acknowledge cycle is complete.

The PCI Slave Channel is done under the following conditions:

- when the TXFIFO is empty (the TXFE bit is set in the MISC_STAT register, Table A-61),
- when the maximum number of bytes per PCI Slave Channel tenure has been reached (as programmed with the PWON field in the MAST_CTL register, Table A-59),
- coupled cycle is complete and the Coupled Window Timer has expired,
- the Coupled Request Timer expires before a coupled cycle is retried by a PCI master, or
- when VMEbus ownership is acquired with the VOWN bit and then the VMEbus ownership bit cleared (see "Exclusive Accesses" on page 2-41).

The DMA Channel is done under the following conditions (see "FIFO Operation and Bus Ownership" on page 2-86 and "DMA Error Handling" on page 2-90):

- DMAFIFO full during VMEbus to PCI bus transfers,
- DMAFIFO empty during PCI bus to VMEbus transfers,
- if an error is encountered during the DMA operation,
- the DMA VMEbus Tenure Byte Counter has expired, or
- DMA block is complete.

The Universe does not monitor BCLR* and so its ownership of the VMEbus is not affected by the assertion of BCLR*.

2.4.2 Universe as VME Master

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 (see “Exclusive Accesses” on page 2-41) and in its role in VMEbus interrupt handling (see “VMEbus Interrupt Handling” on page 2-66).

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.

The Universe 1 chip is unable to perform a 64-bit (double type) floating point operation on a read from VMWbus address, e.g., T=*Vme+1.0. The work around for the Universe 1 chip is to perform the task in two operations, e.g., T=*Vme and T=T+1.0.

2.4.2.1 Addressing Capabilities

Depending upon the programming of the PCI slave image (see “PCI Bus Slave Images” on page 2-47), 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 2-47) 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 (see “RMW and ADOH Cycles” on page 2-40).

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 (Table A-62). After power-up, the two values in the USER_AM register default to the same VME64 user-defined AM code.

2.4.2.2 Data Transfer Capabilities

The data transfer between the PCI bus and VMEbus is perhaps best explained by Figure 2-4 on page 2-14. 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.

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 2-47). 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 (Table A-59). 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.

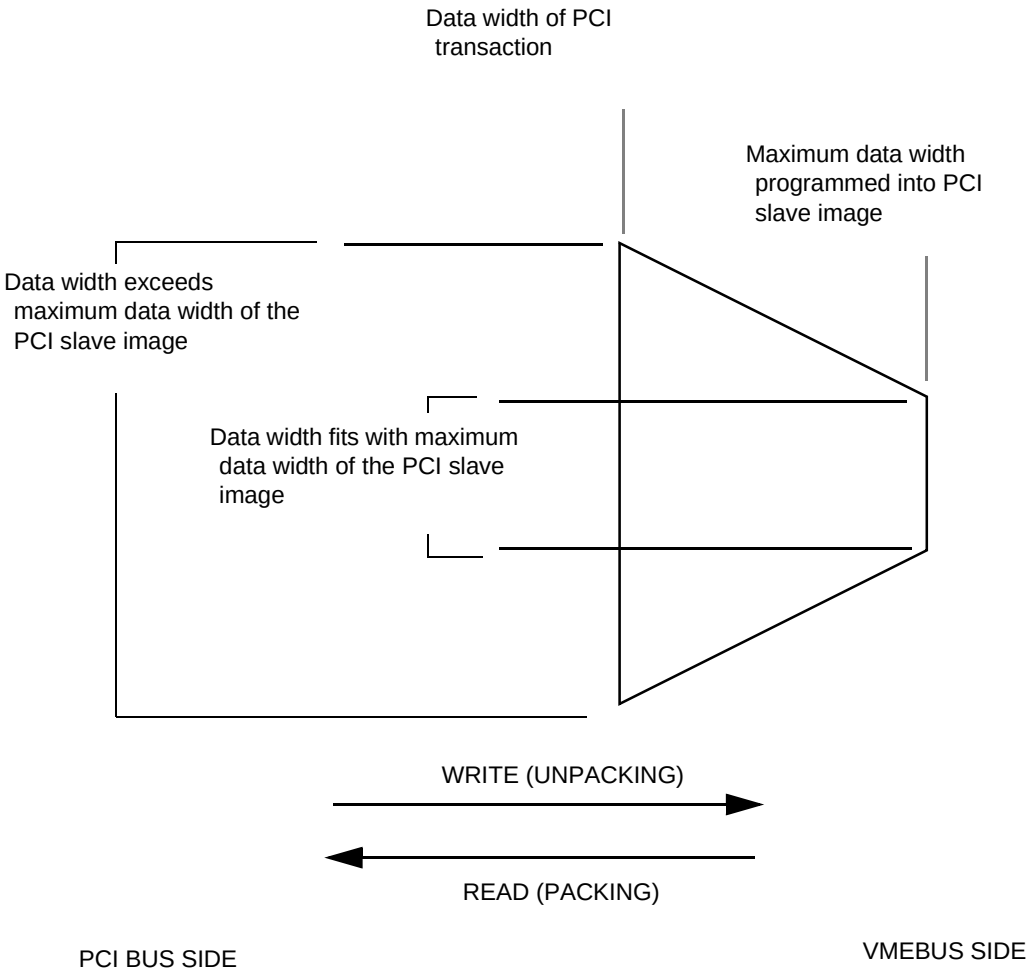


Figure 2-4 Influence of Transaction Data Width and Slave Image Data Width on Data Packing/Unpacking

The Universe provides indivisible transactions with the VMEbus lock commands and the VMEbus ownership bit (see “Exclusive Accesses” on page 2-41).

2.4.2.3 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-83) 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-84). 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 Universe Interrupt Channel” on page 2-58, Table A-46 and Table A-47).

DTACK signals the successful completion of the transaction.

2.4.3 Universe as VME Slave

The Universe becomes 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 2-44 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 2-5 below and “VME Slave Images” on page 2-44). 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).

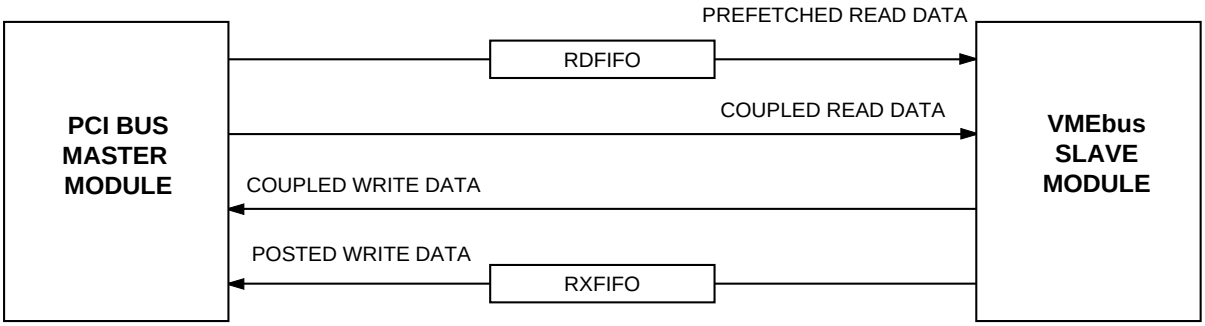


Figure 2-5 VMEbus Slave Channel Dataflow

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.

Note

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-6).

The VMIVME-7589 interface option supports only a 32-bit PCI bus.

2.4.3.1 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” on page 2-44). 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 (Table A-6) when it receives a target abort or master abort.

2.4.3.2 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 2-44). 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 2-44). 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 “Universe as PCI Slave” on page 2-33), 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 (Table A-34) 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-35). An interrupt is generated on the VMEbus and / or PCI bus depending upon whether the VERR and LERR interrupts are enabled (see “Bus Error Handling” on page 2-53 and “The Universe Interrupt Channel” on page 2-58).

2.4.3.3 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 2-44). 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 2-44), 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 2-55 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-63) 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-59). 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*.

The VMIVME-7589 interface option supports only a 32-bit PCI bus.

2.4.3.4 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.

2.4.3.5 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-63), 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.

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.

2.4.3.6 Register Accesses

See "Registers" on page 2-92 for a full description of register mapping and register access.

2.4.3.7 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-60) and is enabled by default.

2.4.4 VMEbus Configuration - First Slot Detector

The Universe provides the First Slot Detector function to assist in the initial configuration of the VMEbus system. As specified by the VME64 specification the First Slot Detector module on the Universe samples BG3IN* immediately after reset to determine whether the Universe's host board resides in slot 1. The VMEbus specification requires that BG[3:0]* lines be driven high after reset. This means that if a card is preceded by another card in the VMEbus system, it will always sample BG3IN* high after reset. BG3IN* can only be sampled low after reset by the first card in the system (there is no preceding card to drive BG3IN* high). If BG3IN* is sampled at logic low immediately after reset (due to the Universe's internal pull-down), then the Universe's host board is in slot 1 and the Universe becomes SYSCON: otherwise, the SYSCON module is disabled. This mechanism may be overridden by software through clearing or setting the SYSCON bit in the MISC_CTL register (Table A-60).

2.4.5 System Controller Functions

When located in Slot 1 of the VME system (see "VMEbus Configuration - First Slot Detector" on page 2-22), the Universe assumes the role of SYSCON and sets the SYSCON status bit in the MISC_CTL register (Table A-60). In accordance with the VME64 specification, as SYSCON the Universe provides:

- a system clock driver,
- an arbitration module,
- an IACK Daisy Chain Driver (DCD), and
- a bus timer.

2.4.5.1 System Clock Driver

The Universe provides a 16 MHz SYSCLK signal derived from CLK64 when configured as SYSCON.

2.4.5.2 VMEbus Arbiter

When the Universe is SYSCON, the Arbitration Module is enabled. The Arbitration Module supports the following arbitration modes:

- Fixed Priority Arbitration Mode (PRI),
- Single Level Arbitration (SGL) (a subset of PRI), or
- Round Robin Arbitration Mode (RRS) (default setting).

These are set with the VARB bit in the MISC_CTL register (Table A-60).

Fixed Priority Arbitration Mode (PRI)

In this mode, the order of priority is VRBR3#, VRBR2#, VRBR1#, and VRBR0# as defined by the VME64 specification. The Arbitration Module issues a Bus Grant (VBGO [3:0]#) to the highest requesting level.

If a Bus Request of higher priority than the current bus owner becomes asserted, the Arbitration Module asserts VBCLR# until the owner releases the bus (VRBBSY# is negated).

Single Level Arbitration Mode (SGL)

In this mode, a subset of priority mode, all requests and grants are made exclusively on level 3. Set the Universe in PRI mode to use this mode.

Round Robin Arbitration Mode (RRS)

This mode arbitrates all levels in a round robin mode, repeatedly scanning from levels 3 to 0. Only one grant is issued per level and one owner is never forced from the bus in favor of another requester (VBCLR# is never asserted).

Since only one grant is issued per level on each round robin cycle, several scans will be required to service a queue of requests at one level.

VMEbus Arbiter Time-out

The Universe's VMEbus arbiter can be programmed to time-out if the requester does not assert BBSY* within a specified period. This allows BGOUT to be negated so that the arbiter may continue with other requesters. The timer is programmed using the VARBTO field in the MISC_CTL register (Table A-60), and can be set to 16 µs, 256 µs, or disabled. The default setting for the timer is 16 µs.

2.4.5.3 IACK Daisy-Chain Driver Module

The IACK Daisy-Chain Driver module is enabled when the Universe becomes system controller. This module guarantees that IACKIN* will stay high for at least 30ns as specified in rule 40 of the VME64 specification.

2.4.5.4 VMEbus Time-out

A programmable bus timer allows users to select a VMEbus time-out period. The time-out period is programmed through the VBTO field in the MISC_CTL register (Table A-60) and can be set to 16µs, 32µs, 64µs, 128 µs, 256 µs, 512 µs, 1024 µs, or disabled. The default setting for the timer is 64 µs. The VMEbus Timer module asserts VXBERR# if a VMEbus transaction times out (indicated by one of the VMEbus data strobes remaining asserted beyond the time-out period).

The Universe timer functions only when the interface is the system controller. The interface provides a non-slot 1 bus timeout timer. Refer to Chapter 3 for additional timer information.

2.4.6 BI-Mode®

BI-Mode (Bus Isolation Mode) is a mechanism for logically isolating the Universe from the VMEbus. Entering BI-mode disables communication between the PCI bus and VMEbus, pointing all transceivers inwards. Interrupt and arbitration daisy chains as well as all SYSCON functions remain operative, but the Universe ceases all VMEbus master and most slave activity. The VMEbus slave images are disabled, but the CR/CSR and UCSRA images are accessible. A PCI master attempting to access a VMEbus resource will be retried. By isolating boards from the VMEbus, the user can implement:

- hot-standby systems
- system diagnostics for routine maintenance, or
- fault isolation in the event of a card failure.

There are three ways to cause the Universe to enter BI-Mode. The Universe is put into BI-Mode if:

1. the BI-mode power-up option is selected (See “Power-up Option Descriptions” on page 2-103 and Table 2-22 on page 2-104). If the Universe is powered-up in BI-mode, then (a) the BI bit in the MISC_CTL register (Table A-60) is set, and (b) any future SYSRST* or PWRRST# will restore the Universe to BI-Mode,
2. the BI bit in the MISC_CTL register is set, or
3. VRIRQ# [1] is asserted, provided that the ENGBI bit in the MISC_CTL register has been set.

The only way to remove the Universe from BI-mode is to clear the BI bit in the MISC_CTL register, provided that the source of the BI-mode is no longer active (i.e., if VRIRQ# [1] is still being asserted while the ENGBI bit in the MISC_CTL register is set, then attempting to clear the BI bit in the MISC_CTL register will not be effective.) Furthermore, as mentioned above, if the Universe has been powered-up in BI-mode, any future SYSRST* or PWRRST# will restore the Universe to BI-Mode.

The BI-Mode power-up option is not supported with the VMIVME-7589 interface option.

2.5 PCI Bus Interface

The PCI Bus Interface is organized as follows:

- "PCI Cycles - Overview" below,
- "Universe as PCI Master" on page 2-30, and
- "Universe as PCI Slave" on page 2-33.

The Universe PCI bus interface is electrically and logically directly connected to the PCI bus. For information concerning the different types of PCI accesses available, see "PCI Bus Slave Images" on page 2-47.

2.5.1 PCI Cycles - Overview

The PCI bus port of the Universe operates as a PCI compliant port with a 64-bit multiplexed address / data bus. The Universe PCI bus interface is configured as little-endian using address invariant translation when mapping between the VMEbus and the PCI bus. Address invariant translation preserves the byte ordering of a data structure in a little-endian memory map and a big-endian memory map.

The Universe has all the PCI signals described in the PCI specification with the exception of SBO# and SDONE (since the Universe does not provide cache support).

Universe PCI cycles are synchronous, meaning that bus and control input signals are externally synchronized to the PCI clock (CLK). PCI cycles are divided into four phases:

1. request,
2. address phase,
3. data transfer, and
4. cycle termination.

2.5.1.1 32-Bit Versus 64-Bit PCI

At reset, the Universe could be configured with a 32-bit or 64-bit PCI data bus, depending on the level of REQ64#.

The VMIVME-7589 Universe based VME interface has been configured for 32-bit PCI interface and does not support 64-bit PCI interface. No attempt should be made to perform 64-bit PCI access.

Each of the Universe's VMEbus slave images can be programmed so that VMEbus transactions are mapped to a 64-bit data bus on the PCI interface (with the LD64EN bit, e.g. see Table A-63).

No attempt should be made to perform 64-bit PCI access. Neither the VMIVME-7589 nor the Universe based VME interface support 64-bit PCI transfers.

If the Universe VME slave images are not programmed for a 64-bit wide PCI data bus, then the Universe operates transparently in a 32-bit PCI environment.

The VMIVME-7589 interface option supports only a 32-bit PCI bus.

2.5.1.2 PCI Bus Request

When the Universe requires control of the PCI bus it asserts REQ# and gains bus mastership when the PCI arbiter asserts GNT#. GNT# assertion allows bus access for a single transaction. If the arbiter removes GNT# after the Universe has begun its PCI transaction, the Universe completes the current transaction and releases the PCI bus (removes REQ#). This means that the Universe PCI Master Interface may have to re-arbitrate for the PCI bus after every transaction if its GNT# is removed.

2.5.1.3 Address Phase

PCI transactions are initiated by asserting FRAME# and driving address and command information onto the bus. In the VMEbus Slave Channel, the Universe calculates the address for the PCI transaction by adding a translation offset to the VMEbus address (see "Universe as VME Slave" on page 2-15).

The command signals (on the C/BE# lines) contain information about memory space, cycle type and whether the transaction is read or write. Table 2-2 below gives PCI the command type encoding implemented with the Universe.

Memory Read Multiple and Memory Read Line transactions are aliased to Memory Read transactions when the Universe is accessed as a PCI target with these commands. Likewise, Memory Write and Invalidate is aliased to Memory Write. As a PCI initiator, the Universe never generates these commands.

Table 2-2 Command Type Encoding for Transfer Type

C/BE# [3:0] for PCI, C/BE# [7:4] for non-multiplexed	Command Type	Universe Capability
0000	Interrupt Acknowledge	N / A
0001	Special Cycle	N / A
0010	I / O Read	Slave / Master
0011	I / O Write	Slave Master
0100	Reserved	N / A
0101	Reserved	N / A
0110	Memory Read	Slave / Master
0111	Memory Write	Slave / Master
1000	Reserved	N / A
1001	Reserved	N / A
1010	Configuration Read	Slave / Master
1011	Configuration Write	Slave / Master
1100	Memory Read Multiple	(See Text)
1101	Dual Address Cycle	N / A
1110	Memory Read Line	(See Text)
1111	Memory Write and Invalidate	(See Text)

PCI slaves are expected to assert DEVSEL# if they have decoded the access. During a Configuration cycle, the slave is selected by its particular IDSEL#. If a target does not respond with DEVSEL# within 6 clocks, a master abort is generated. The role of configuration cycles is described in the PCI 2.1 Specification.

2.5.1.4 Data Transfer

The acknowledgment for a data phase for a transaction occurs on the first rising clock edge after both IRDY# and TRDY# are asserted by the master and slave, respectively. REQ64# may be driven during the address phase to indicate that the master wishes to initiate a 64-bit transaction. The PCI slave asserts ACK64# if it is able to respond to the 64-bit transaction.

Wait cycles are introduced by either the master or the slave by deasserting IRDY# or TRDY#. For write cycles, data is valid on the first rising edge after IRDY# is asserted. Data is acknowledged by the slave on the first rising edge with TRDY# asserted. For read cycles, data is transferred and acknowledged on first rising edge with both IRDY# and TRDY# asserted.

A single data transfer cycle is repeated every time IRDY# and TRDY# are both asserted. The transaction only enters the termination phase when FRAME# is deasserted (master initiated termination) or if STOP# is asserted (slave initiated). When both FRAME# and IRDY# are deasserted (final data phase is complete), the bus is defined as idle.

2.5.1.5 Termination Phase

The Universe PCI bus interface permits all four types of PCI terminations:

1. Master-Abort: the PCI bus master negates FRAME# when no slave responds (DEVSEL# not asserted) after 6 clock cycles.
2. Target-Disconnect: a termination is requested by the slave (STOP# is asserted) because it is unable to respond within the latency requirements of the PCI specification or it requires a new address phase. Target-disconnect means that the transaction is terminated after data is transferred. The Universe will deassert REQ# for at least two clock cycles if it receives STOP# from the PCI slave.
3. Target-Retry: termination is requested (STOP# is asserted) by the slave because it cannot currently process the transaction. Retry means that the transaction is terminated after the address phase without any data transfer.
4. Target-Abort: is a modified version of target-disconnect where the slave requests a termination (asserts STOP#) of a transaction which it will never be able to respond to, or during which a fatal error occurred. Although there may be a fatal error for the initiating application, the transaction completes gracefully, ensuring normal PCI operation for other PCI resources.

2.5.1.6 Parity Checking

The Universe both monitors and generates parity information on the PAR# signal. The Universe monitors PAR# when it accepts data as a master during a read or a slave during a write. The Universe drives PAR# when it provides data as a slave during a read or a master during a write. The Universe also drives PAR# during the address phase of a transaction when it is a master and monitors PAR# during an address phase when it is the PCI slave. In both address and data phases, the PAR# signal provides even parity for C/BE#[7:0] and AD[63:0]. The Universe continues with a transaction independent of any parity errors reported during the transaction.

The Universe can also be programmed to report address parity errors. It does this by asserting the SERR# signal and setting a status bit in its registers. No interrupt is generated, and regardless of whether assertion of SERR# is enabled, the Universe does not respond to the errored access. If powered up in a 64-bit PCI environment, the Universe uses PAR64 in the same way as PAR, except for AD[63:32] and C/BE[7:4].

2.5.2 Universe as PCI Master

The Universe requests PCI bus mastership through its PCI Master Interface. The PCI Master Interface is available to either the VME Slave Channel (access from a remote VMEbus master) or the DMA Channel.

The VME Slave Channel makes an internal request for the PCI Master Interface when:

- the RXFIFO contains a complete transaction,
- sufficient data exists in the RXFIFO to generate a transaction of length defined by the programmable aligned burst size (PABS), or
- there is a coupled cycle request.

The DMA Channel makes an internal request for the PCI Master Interface when:

- the DMAFIFO has sufficient room to queue an entire transaction (of length defined by PABS) during reads from the PCI bus
- the DMAFIFO has queued an entire transaction (of length defined by PABS) and is ready to write to the PCI bus, or
- the DMA block is completely queued during a write to the PCI bus.

Arbitration between the two channels for the PCI Master Interface follows a round robin protocol. Each channel is given access to the PCI bus for a single transaction. Once that transaction completes, ownership of the PCI Master Interface is granted to the other channel if it requires the bus. The VME Slave Channel and the DMA Channel each have a set of rules that determine when it is ‘done’ with the PCI Master Interface. The VME Slave Channel is done under the following conditions:

- an entire transaction (no greater in length than the programmed aligned burst size) is emptied from the RXFIFO, or
- the coupled cycle is complete.

The DMA Channel is done when:

- the boundary programmed into the PCI aligned burst size is emptied from the DMAFIFO during writes to the PCI bus, or
- the boundary programmed into the PCI aligned burst size is queued to the DMAFIFO during reads from the PCI bus.

As discussed elsewhere (“Universe as VME Slave” on page 2-15), access from the VMEbus may be either coupled or decoupled. For a full description of the operation of these data paths, see “Universe as VME Slave” on page 2-15.

The Universe PCI Master Interface can generate the following command types:

- I/O Read,
- I/O Write,
- Memory Read,
- Memory Write,
- Configuration Read (Type 0 and 1), and
- Configuration Write (Type 0 and 1).

The type of cycle the Universe generates on the PCI bus depends upon which VMEbus slave image is accessed and how it is programmed. For example, one slave image might be programmed as an I/O space and another as memory space (see “VME Slave Images” on page 2-44). When generating a memory transaction, the implied addressing is either 32-bit or 64-bit aligned, depending upon the PCI slave. When generating an I/O transaction, the implied addressing is 32-bit aligned and all incoming transactions are coupled.

Both Type 0 and Type 1 cycles are generated and handled through the same mechanism. Once a VME cycle is received and mapped to a configuration cycle, the Universe compares bits [23:16] of the incoming address with the value stored in the Universe’s MAST_CTL Register’s Bus Number field (BUS_NO[7:0] in Table A-59). If the bits are the same as the BUS_NO field, then a TYPE 0 access is generated. If they are not the same, a Type 1 configuration access is generated. The PCI bus-generated address then becomes an unsigned addition of the incoming VMEbus address and the VME slave image translation offset.

2.5.2.1 PCI Burst Transfers

The Universe generates aligned burst transfers of some maximum alignment, according to the programmed PCI aligned burst size. The PCI aligned burst size can be programmed at 32 or 64 bytes. Burst transfers will not cross 32-byte or 64-byte boundaries, depending on the programmed alignment (PABS bit in the MAST_CTL register, Table A-59). For example, when programmed for 32-byte boundaries, a new burst will begin at XXXX_XX20, XXXX_XX40, etc. If necessary, a new burst will begin at an address with the programmed alignment. To optimize PCI bus usage, the Universe always attempts to transfer data in aligned bursts at the full negotiated width of the PCI bus.

The Universe can perform a 64-bit data transfer over the AD [63:0] lines, if operated in a 64-bit PCI environment or against a 64-bit capable target or initiator. The LD64EN bit must be set if the access is being made through a VMEbus slave image; the LD64EN bit must be set if the access is being performed with the DMA.

The Universe generates burst cycles on the PCI bus if it is:

- emptying the RXFIFO (the RXFE status bit in the MISC_STAT register is set when the RXFIFO empties),

- filling the RDFIFO (receives a block read request from a VMEbus master to an appropriately programmed VMEbus slave image), or
- performing DMA transfers

All other accesses are treated as single data beat transactions on the PCI bus.

During PCI burst transactions, the Universe dynamically enables byte lanes on the PCI bus by changing the BE# signals during each data phase.

2.5.2.2 Termination

The Universe performs a master abort if the slave does not respond within 6 clock cycles. Coupled PCI transactions terminated with target abort or master abort are terminated on the VMEbus with BERR*. The R_TA or R_MA bits in the PCI_CS register (Table A-6) are set when the Universe receives a target abort or generates a master abort independent of whether the transaction was coupled, decoupled, prefetched, or initiated by the DMA.

If the Universe receives a retry from the PCI slave, then it relinquishes the PCI bus and re-requests within 2-3 PCI clock cycles. No other transactions are processed by the PCI Master Interface until the retry condition is cleared. The Universe can be programmed to perform a maximum number of retries using the MAXRTRY field in the MAST_CTL register (Table A-59). When this number of retries has been reached, the Universe responds in the same way as it does to a target abort on the PCI bus. That is, the Universe may issue a BERR* signal on the VMEbus. Target aborts are discussed in the next two paragraphs. All VME slave coupled transactions and decoupled transactions will encounter a delayed DTACK once the FIFO fills until the condition clears either due to success or a retry time-out.

If the error occurs during a posted write to the PCI bus (see also “Bus Error Handling” on page 2-53), the Universe uses the L_CMDERR register (Table A-34) to log the command information for the transaction (CMDERR [3:0]) and the address of the errored transaction is latched in the LAERR register (Table A-35). The L_CMDERR register also records if multiple errors occur (with the M_ERR bit) although the number of errors is not given. The error log is qualified with the L_STAT bit. The rest of the transaction will be purged from the RXFIFO if some portion of the write encounters an error. An interrupt is generated on the VMEbus and/or PCI bus depending upon whether the VERR and LERR interrupts are enabled (see “The Universe Interrupt Channel” on page 2-58).

If an error occurs during a prefetched read, it is only translated to the VMEbus (as a BERR* signal) during the first data beat (the coupled portion) of the prefetched read. 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.

2.5.2.3 Parity

The Universe monitors PAR# when it accepts data as a master during a read and drives PAR# when it provides data as a master during a write. The Universe also drives PAR# during the address phase of a transaction when it is a master. In both address and data phases, the PAR# signal provides even parity for C/BE#[3:0] and AD[31:0]. If the Universe is powered up in a 64-bit PCI environment, then PAR64 provides even parity for C/BE#[7:4] and AD[63:32].

The PERESP bit in the PCI_CS register (Table A-6) determines whether or not the Universe responds to parity errors as PCI master. Data parity errors are reported through the assertion of PERR# if the PERESP bit is set. Regardless of the setting of these two bits, the D_PE (Detected Parity Error) bit in the PCI_CS register is set if the Universe encounters a parity error as a master. The DP_D (Data Parity Detected) bit in the same register is only set if parity checking is enabled through the PERESP bit and the Universe detects a parity error while it is PCI master (i.e. it asserts PERR# during a read transaction or receives PERR# during a write).

No interrupts are generated by the Universe in response to parity errors reported during a transaction. Parity errors are reported by the Universe through assertion of PERR# and by setting the appropriate bits in the PCI_CS register. If PERR# is asserted to the Universe while it is PCI master, the only action it takes is to set the DP_D. The Universe continues with a transaction independent of any parity errors reported during the transaction.

As a master, the Universe does not monitor SERR#. It is expected that a central resource on the PCI bus will monitor SERR# and take appropriate action.

2.5.3 Universe as PCI Slave

The Universe becomes PCI bus slave when one of its four programmed PCI slave images or one of its registers is accessed by a PCI bus master (the Universe cannot be that PCI bus master). Register accesses are discussed elsewhere (see “Registers” on page 2-92). This section describes only those accesses destined for the VMEbus.

When one of its PCI slave images is accessed, the Universe responds with DEVSEL# within two clocks of FRAME# (making the Universe a medium speed device, as reflected by the DEVSEL field in the PCI_CS register).

As PCI slave, the Universe responds to the following command types:

- I/O Read,

- I/O Write,
- Memory Read,
- Memory Write,
- Configuration Read (Type 1),
- Configuration Write (Type 1),
- Memory Read Multiple (aliased to Memory Read),
- Memory Line Read (aliased to Memory Read),
- Memory Write and Invalidate (aliased to Memory Write).

Type 0 configuration accesses can only be made to the Universe's PCI configuration registers. The PCI slave images do not accept Type 0 accesses; they only accept Type 1 accesses if AD[23:16] matches the BUS_NO field in the MAST_CTL register.

Address parity errors are reported if both PERESP and SERR_EN are set in the PCI_CS register (Table A-6). Address parity errors are reported by the Universe by asserting the SERR# signal and setting the S_SERR (Signalled SERR#) bit in the PCI_CS register. Assertion of SERR# can be disabled by clearing the SERR_EN bit in the PCI_CS register. No interrupt is generated, and regardless of whether assertion of SERR# is enabled or not, the Universe does not respond to the access with DEVSEL#. Typically the master of the transaction times out with a master-abort.

If the Universe is accessed with REQ64# as a 64-bit target, then it responds with ACK64# if it is powered up as a 64-bit device.

If the VME Cycle Type field (VCT) of the PCI Slave Image Control Register (LSIx_CTL in Table A-11 or Table A-15) corresponding to the incoming AD signal is set to binary 10, then the incoming AD[23:16] is compared to the BUS_NO field in the MAST_CTL Register (Table A-59). If they match then the cycle is accepted and a VMEbus cycle is generated.

2.5.3.1 Data Transfer

Read transactions are always coupled (as opposed to VMEbus slave reads, which may be pre-fetched; see "Universe as VME Slave" on page 2-15). Write transactions can be coupled or posted (see Figure 2-6 below and "PCI Bus Slave Images" on page 2-47). To ensure sequential consistency, coupled operations (reads or writes) are only processed once all previously posted write operations have completed (i.e. the TXFIFO is empty).

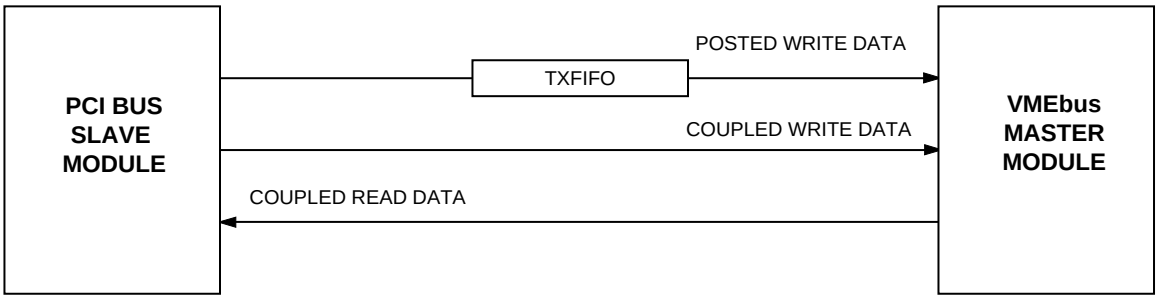


Figure 2-6 PCI bus Slave Channel Dataflow

The data transfer between the PCI bus and VMEbus is perhaps best explained by Figure 2-7 below. 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 (VDW bit in the PCI slave image control register). 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.

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.

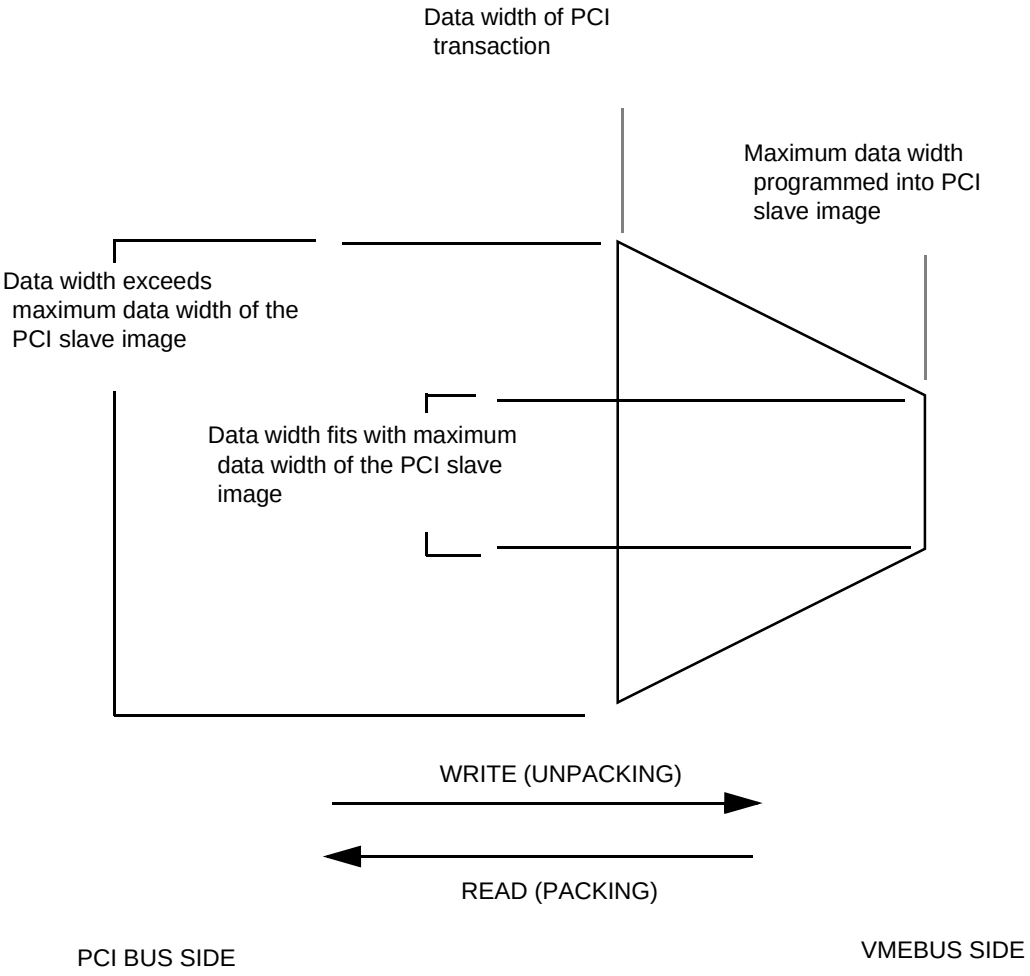


Figure 2-7 Influence of Transaction Data Width and Slave Image Data Width on Data Packing/Unpacking

2.5.3.2 Coupled Transfers

By default, all PCI slave images are set for coupled transfers. If a data phase of a coupled burst requires packing or unpacking on the VMEbus, acknowledgment of the transfer is not received on the PCI bus until all data has been packed or unpacked on the VMEbus. If an error occurs during any of the packing or unpacking, then the transaction is terminated on the PCI bus with Target-Abort. Target retries will be generated by the Universe while there are posted writes in the TXFIFO or while the PCI Slave Channel does not have VMEbus ownership. Note that the Universe may be the owner of the VMEbus through the DMA or Interrupt Channels in which case the PCI master will be retried. See “VMEbus Release” on page 2-10 for more details on how the Universe shares the VMEbus between channels.

PCI latency requirements (as described in revision 2.0 of the PCI Specification) require that only 8 clock cycles can elapse between the first and second data beat of a transaction. Since the Universe cannot guarantee that data acknowledgment will be received from the VMEbus in time to meet these PCI latency requirements, the Universe performs a target-disconnect after the first data beat of every coupled transaction.

Retried Coupled Transfers

As PCI slave, the Universe retries all coupled VMEbus accesses unless the Local Slave Channel is the current owner of the VMEbus. This prevents deadlock conditions and improves the utilization of the PCI bus (i.e. increases the probability that a coupled transfer will proceed as soon as possible without blocking all other PCI bus traffic).

After initiating a target-retry to a coupled access, the Universe requests the VMEbus. While VMEbus ownership is pending, it continues to retry all PCI masters, both coupled and decoupled, except for those accessing the Universe registers. Once VMEbus ownership is obtained, any coupled request from a PCI master is mapped directly to the VMEbus.

The Universe has a configurable Coupled Request Timer (set with the CRT field in the LMISC register, Table A-32) that is restarted every time a coupled request is retried. The CRT can be programmed to 128 µs, 256 µs, 512 µs, 1024 µs, 2048 µs, 4096 µs, or disabled. By default, the CRT is set to 128 µs. If set to 0, the VMEbus is held until a coupled request arrives. If the Universe obtains the VMEbus and receives no request for coupled access before the Coupled Request Timer expires, then it will release the VMEbus.

Consecutive Coupled Transfers

The Universe has a programmable coupled window timer that allows consecutive coupled transactions to occur without the need to release the VMEbus between transactions. After completing a coupled transaction, the Universe delays its release of the VMEbus by the amount of time programmed into the coupled window timer. The timer can be set with the CWT field in the LMISC register (Table A-32) to count PCI

clocks. It can be programmed to 16, 32, 64, 128, 256, or 512 clocks. By default the CWT is disabled so that VMEbus ownership is relinquished immediately by the PCI Slave Channel after each coupled cycle. If the release mode is set for RWD or there is an external VME request pending in ROR mode, then this causes VMEbus ownership to be relinquished by the Universe independent of other channel requests.

If another coupled transaction is generated during the timer period, it will be processed directly and will not be retried. If another cycle is not attempted before the window expires, then the PCI Slave Channel releases its ownership of the VMEbus.

2.5.3.3 Posted Writes

Posted writes are enabled for a PCI slave image by setting the PWEN bit in the control register of the PCI Slave Image (see “PCI Bus Slave Images” on page 2-47). Write transactions are relayed from the PCI bus to the VMEbus through a 18-entry deep TXFIFO. The TXFIFO allows each entry to contain 32 address bits (with extra bits provided for command information), or up to 64 data bits. For each posted write transaction received from the PCI bus, the PCI Slave Interface queues an address entry in the FIFO. This entry contains the translated address space and mapped VMEbus attributes information relevant to the particular PCI slave image that has been accessed (see “PCI Bus Slave Images” on page 2-47). For this reason, any re-programming of PCI bus slave image attributes will only be reflected in TXFIFO entries queued after the re-programming. Transactions queued before the re-programming are delivered to the VMEbus with the PCI bus slave image attributes that were in use before the re-programming.

Care should be taken before reprogramming slave images from one bus while that image is being accessed from the opposite bus. If there is a chance the image may be accessed while being reprogrammed, disable the image first before changing image attributes.

Once the address phase is queued in one TXFIFO entry, the PCI Slave Interface may pack the subsequent data beats to a full 64-byte width before queuing the data into new entries in the TXFIFO.

The number of data beats that the PCI Slave Channel queues in the TXFIFO for a single posted write depends upon the programming of the PCI aligned burst size field (PABS) in the MAST_CTL register (Table A-59). The PABS field is programmable to 32 or 64 bytes (coupled transactions are not affected by the PABS field and are always broken up after the first data beat, see above). When a PCI master attempts a posted

write transaction that crosses the programmed aligned burst boundary, the Universe's PCI Slave Interface generates a target disconnect to the master at the aligned burst boundary. The transaction up to the aligned burst boundary is queued in the TXFIFO as one address entry plus up to four or eight data entries (32 or 64 bytes) depending on the programming of PABS.

To increase performance and bus utilization on the PCI bus, the Universe does not queue a transaction into the TXFIFO until the TXFIFO contains enough space for a full aligned burst (as defined by PABS). If the TXFIFO does not have enough space for an aligned burst, then the PCI posted write transaction is terminated by the Universe with a target-retry immediately after the address phase. This mechanism, where the PCI Slave Interface only accepts transactions when it has sufficient room for a full aligned burst, ensures that the Universe always accepts a full transaction (as defined by the programmed aligned burst size) without inserting wait states or terminating a transaction early.

Additionally, although there are 18 entries available in the Tx FIFO, a maximum of four complete transactions may be queued at any one time. If four transaction have already been queued, the PCI Slave Interface retries all subsequent accesses until a transaction has been delivered to the VMEbus.

Before a transaction can be delivered to the VMEbus from the TXFIFO, the PCI Slave Channel must obtain ownership of the VME Master Interface. Ownership of the VME Master Interface is granted to the different channels on a round robin basis (see "VMEbus Release" on page 2-10). Once the PCI Slave Channel obtains the VMEbus through the VME Master Interface, the manner in which the TXFIFO entries are delivered depends upon the programming of the VMEbus attributes in the PCI slave image (see "PCI Bus Slave Images" on page 2-47). For example, if the VMEbus data width is programmed to 16 bits, and block transfers are disabled, then each data entry in the TXFIFO corresponds to four transactions on the VMEbus.

If block transfers are enabled in the slave image, BLTs or MBLTs are generated on the VMEbus such that each transaction queued in the TXFIFO corresponds to a single VMEbus block transfer. This means that unlike the DMA Channel (which can generate full block sizes of 256 bytes for BLTs and 2 Kbytes for MBLTs), the PCI Slave Channel can generate block transfers only at the programmed aligned burst size (32 or 64 bytes, PABS in the MAST_CTL register, Table A-59). This corresponds (depending upon the programming of PABS) to eight or sixteen data phases per BLT; four or eight data phases per MBLT. In addition, note that if block transfers are enabled in the PCI slave image, then each transaction queued in the TXFIFO, independent of its length, is delivered to the VMEbus as a block transfer. This means that if a single data beat transaction is queued in the TXFIFO, it appears on the VMEbus as a single data phase block transfer.

Any PCI master attempting coupled transactions is retried while the TXFIFO contains data.

2.5.3.4 RMW and ADOH Cycles

The Universe will only generate RMW and ADOH cycles on the VMEbus through the use of a Special Cycle Generator. The Special Cycle Generator is configured through the register fields shown in Table 2-3 below.

Table 2-3 Register Fields for the Special Cycle Generator

Field	Register Bits	Description
32-bit address	ADDR in Table A-28	specifies PCI bus slave image address
type	SCYC in Table A-27	disabled, RMW or ADOH
32-bit enable	EN [31:0] in Table A-29	a bit mask to select the bits to be modified in the VMEbus read data during a RMW cycle
32-bit compare	CMP [31:0] in Table A-30	data which is compared to the VMEbus read data during a RMW cycle
32-bit swap	SWP [31:0] in Table A-31	data which is swapped with the VMEbus read data and written to the original address during a RMW cycle

The 32-bit address field is used to select an address within one of the currently enabled A32 PCI slave images. When the SCYC field is programmed, the VMEbus transaction parameters associated with the selected VMEbus address are temporarily disabled, although the translation offset for the slave image still applies. Instead, the VMEbus transaction parameters used in the transaction are those specified in the Special Cycle Generator registers. When a PCI master accesses the selected VMEbus address while the SCYC field is set, a coupled cycle of the programmed type (RMW or ADOH) is executed on the VMEbus. Either a read or write to the selected address will initiate the RMW or ADOH transaction (the different effects of a read or write are described in the separate RMW and ADOH sections below). When the VMEbus transaction is complete (and the SCYC field is cleared), the selected transaction parameters belonging to the PCI address return to those defined in the PCI slave image.

The following sections describe the specific properties for each of the transfer types: RMW and ADOH.

Read-Modify-Write

When the SCYC field is set to RMW, any read access to the specified VMEbus address will result in a RMW cycle on the VMEbus. The data from the read portion of the RMW on the VMEbus is returned as the read data on the PCI bus. If initiated with a write transaction, a standard write occurs with the attributes programmed in the PCI slave image.

RMW cycles make use of three 32-bit registers. The bit enable field is a bit mask which lets the user specify which bits in the read data are compared and modified in the RMW cycle. This bit enable setting is completely independent of the RMW cycle data width, which is determined by the data width of the initiating PCI transaction. During a RMW, the VMEbus read data is bitwise compared with the SCYC_CMP and SCYC_EN registers. The valid compared and enabled bits are then swapped using the SCYC_SWP register.

Each enabled bit that compares true is swapped with the corresponding bit in the 32-bit swap field. A false comparison results in the original bit being written back.

Once the RMW cycle completes, the VMEbus read data is returned to the waiting PCI bus master and the PCI cycle terminates.

ADOH

When the SCYC field is set to ADOH, any write access to the specified VMEbus address will result in an ADOH cycle on the VMEbus. Reads to the specified address translates to VMEbus reads in the standard fashion. The data during writes is ignored. The AM code generated on the VMEbus is determined by the PCI slave image definition for the specified VMEbus address (see Table 2-11 on page 2-50).

The ADOH cycle is used to implement the VMEbus Lock command. However, after the ADOH cycle is complete, there is no guarantee that the Universe will remain VMEbus master. If the Universe loses VMEbus ownership, then the VMEbus Lock command will no longer apply. The user can use the VMEbus ownership described in “Exclusive Accesses” on page 2-41 to ensure that the Universe will maintain VMEbus ownership after the ADOH cycle is complete.

2.5.3.5 Exclusive Accesses

A PCI master obtains exclusive access to a resource on the VMEbus through the combined use of the PCI LOCK# signal (which locks the PCI Slave Interface against other PCI masters) and the VMEbus ADOH cycle (which locks the VMEbus resource against other VMEbus masters). LOCK# ensures the PCI master sole PCI access to the Universe registers and the VMEbus. The ADOH cycle ensures that the VME Master Interface has exclusive access to the VMEbus resource while it has VMEbus tenure. If there is only one master on the PCI bus, there is no need for the PCI lock.

To further guarantee exclusive access, the Universe provides a VMEbus ownership bit (VOWN bit in the MAST_CTL register, Table A-59) to ensure that the Universe has access to the locked VMEbus resource for an indeterminant period. The Universe can be programmed to assert an interrupt on the PCI bus when it acquires the VMEbus and the VOWN bit is set (VOWN enable bit in the LINT_EN register, Table A-43). While the VMEbus is held using the VOWN bit, the Universe sets the VOWN_ACK bit in the MAST_CTL register. The VME Master Interface maintains bus tenure while the ownership bit is set, and only releases the VMEbus when the ownership bit is cleared. This function is important for the following two reasons.

If the VME Master Interface is programmed for RWD, it may release the VMEbus when the PCI Slave Channel has completed a transaction (definition of 'done' for the PCI Slave Channel, see "VMEbus Release" on page 2-10). Therefore, if exclusive access to the VMEbus resource is required for multiple transactions, then the VMEbus ownership bit will hold the bus until the exclusive access is no longer required.

Alternatively, if the VME Master Interface is programmed for ROR, the VMEbus ownership bit will ensure VMEbus tenure even if other VMEbus requesters require the VMEbus.

Note

Posted writes will be treated as coupled transactions while the VMEbus ownership bit is set.

2.5.3.6 Terminations

The Universe performs the following terminations as PCI slave:

1. Target-Disconnect
   - when registers are accessed (no bursts allowed to registers),
   - after the first data beat of every coupled cycle,
   - when the transaction reaches a programmable aligned burst size boundary.
2. Target-Retry
   - when a PCI master tries to write to the TXFIFO when there is insufficient room for a full burst,
   - when a coupled transaction is attempted and the Universe does not own the VMEbus, or
   - when a coupled transaction is attempted while the TXFIFO has entries to process.
3. Target-Abort
   - when the Universe receives BERR* on the VMEbus during a coupled cycle (BERR* translated as Target-Abort on the PCI side and the S_TA bit is set in the PCI_CS register, Table A-6).

Whether to terminate a transaction or for retry purposes, the Universe keeps STOP# asserted until FRAME# is deasserted, independent of the logic levels of IRDY# and TRDY#. If STOP# is asserted while TRDY# is deasserted, it means that the Universe will not transfer any more data to the master.

If an error occurs during a posted write to the VMEbus, the Universe uses the V_AMERR register (Table A-83) 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 FIFO entries for the offending cycle are purged. The V_AMERR register also records whether multiple errors have occurred (with the

M_ERR bit) although the number is not given. The error log is qualified with the V_STAT bit (logs are valid if the V_STAT bit is set). The address of the errored transaction is latched in the VAERR register (Table A-84). 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 VERR interrupts are enabled (see “The Universe Interrupt Channel” on page 2-58).

2.6 Slave Image Programming

The Universe recognizes two types of accesses on its bus interfaces: accesses destined for the other bus, and accesses decoded for its own register space. Address decoding for the Universe's register space is described in "Registers" on page 2-92. This section describes the slave images used to map transactions between the PCI bus and VMEbus.

2.6.1 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.

Table 2-4 VMEbus Fields for VME Slave Image

Field	Register Bits	Description
base	BS[31:12] in Table A-64	multiples of 4 or 64 Kbytes (base to bound: maximum of 4 GBytes)
bound	BD[31:12] in Table A-65	
address space	VAS in Table A-63	A16, A24, A32, User 1, User 2
mode	SUPER in Table A-63	supervisor and/or non-privileged
type	PGM in Table A-63	program and/or data

Table 2-5 PCI Bus Fields for VME Slave Image

Field	Register Bits	Description
translation offset	TO[31:12] in Table A-66	offsets VMEbus slave address to a selected PCI address
address space	LAS in Table A-63	Memory, I/O, Configuration
RMW	LLRMW in Table A-63	RMW enable bit

Table 2-6 Control Fields for VME Slave Image

Field	Register Bits	Description
image enable	EN in Table A-62	enable bit
posted write	PWEN in Table A-63	posted write enable bit
prefetched read	PREN in Table A-63	prefetched read enable bit **Warning**: Always set to '0'
enable PCI D64	LD64EN in Table A-63	enables 64-bit PCI bus transactions

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.

WARNING

Do not set bit LD64EN to '1' in any of the following registers: VSI0_CTL, VSI1_CTL, VSI2_CTL, and VSI3_CTL. Setting any of these bits will damage the VMIVME-7589.

2.6.1.1 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.

Note

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.

2.6.1.2 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 2-21). 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.

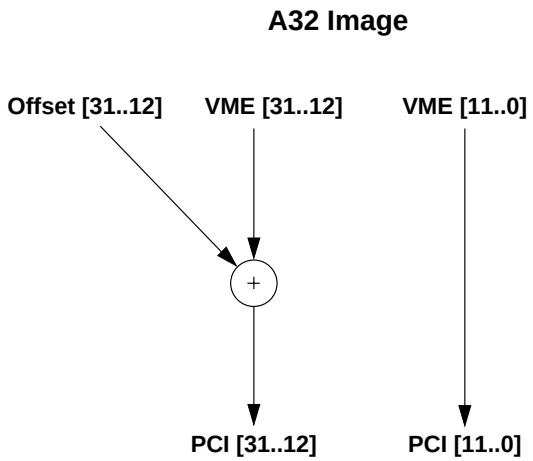


Figure 2-8 Address Translation Mechanism for VMEbus to PCI Bus Transfers

2.6.1.3 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 2-17). 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-6).

2.6.2 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 2-50.

Table 2-7 PCI Bus Fields for the PCI Bus Slave Image

Field	Register Bits	Description
base	BS[31:12] in Table A-12	multiples of 4 or 64 Kbytes (base to bound: maximum of 4 GBytes)
bound	BD[31:12] in Table A-13	
address space	LAS in Table A-13	Memory, I/O, Configuration

Table 2-8 VMEbus Fields for the PCI Bus Slave Image

Field	Register Bits	Description
translation offset	TO[31:12] in Table A-14	translates address supplied by PCI master to a specified VMEbus address
maximum data width	VDW in Table A-11	8, 16, 32, or 64 bits
address space	VAS in Table A-11	A16, A24, A32, CR/CSR, User1, User2
mode	SUPER in Table A-11	supervisor or non-privileged
type	PGM in Table A-11	program or data
cycle	VCT in Table A-11	single or block

Table 2-9 Control Fields for PCI Bus Slave Image

Field	Register Bits	Description
image enable	EN in Table A-11	enable bit
posted write	PWEN in Table A-11	enable bit

Note

If the programming for an image is changed after the transaction is queued in the FIFO, the transaction's attributes are not changed.

2.6.2.1 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

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.

2.6.2.2 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:.

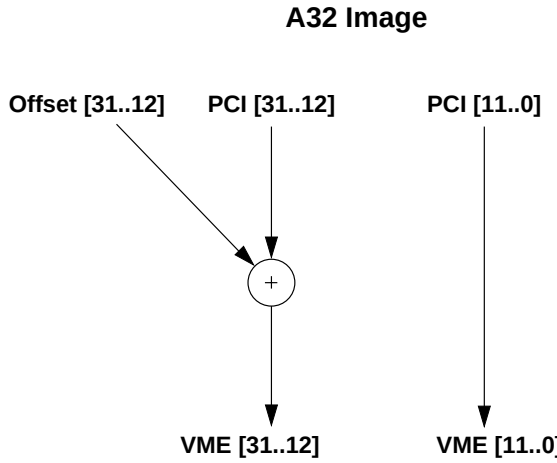


Figure 2-9 Address Translation Mechanism for PCI Bus to VMEbus Transfers

Translations beyond the 4 Gbyte limit will wrap around to the low address range.

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 (see “RMW and ADOH Cycles” on page 2-40).

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 (Table A-62) 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.

2.6.2.3 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.

2.6.3 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 2-10 on page 2-52). 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.

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

Table 2-10 PCI Bus Fields for the Special PCI Slave Image

Field	Register Bits	Description
base	BS[5:0] in Table A-33	64 Mbyte aligned base address for the image
address space	LAS [1:0] in Table A-33	Places image in Memory, I/O, or Configuration space

Table 2-11 VMEbus Fields for the Special PCI Bus Slave Image

Field	Register Bits	Description
maximum data width	VDW in Table A-33	separately sets each region for 16 or 32 bits
mode	SUPER in Table A-33	separately sets each region as supervisor or non-privileged
type	PGM in Table A-33	separately sets each region as program or data

Table 2-12 Control Fields for the Special PCI Bus Slave Image

Field	Register Bits	Description
image enable	EN in Table A-33	enable bit for the image
posted write	PWEN in Table A-33	enable bit for posted writes for the 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 2-47), 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.

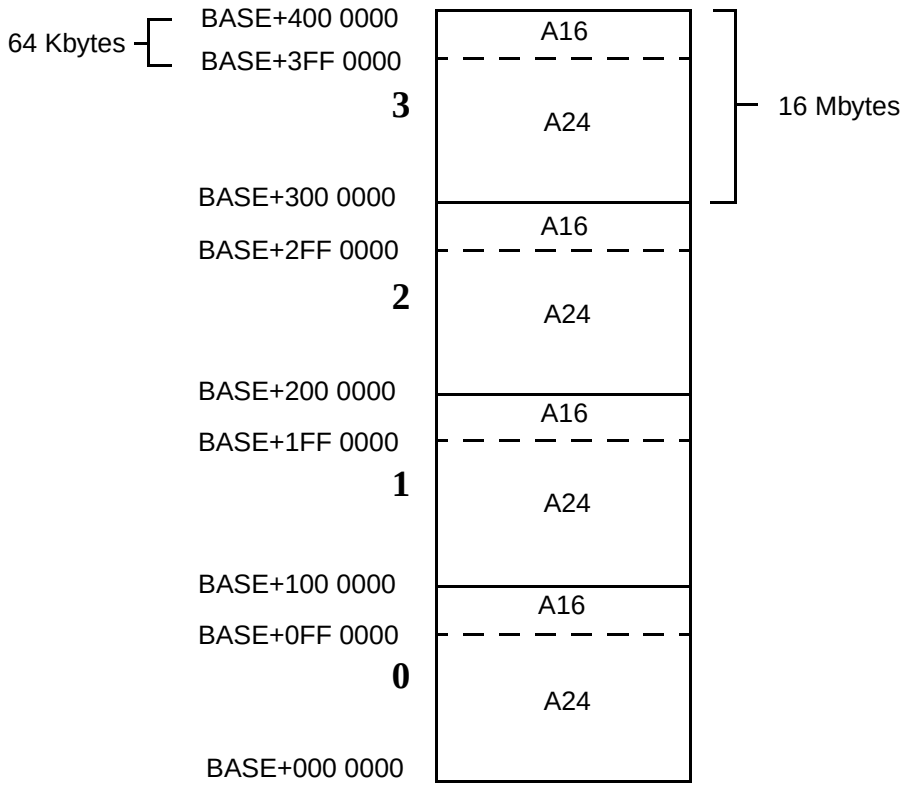


Figure 2-10 Memory Mapping in the Special PCI Slave Image

2.7 Bus Error Handling

There are two fundamentally different conditions under which bus errors may occur with the Universe: during coupled cycles or during decoupled cycles. In a coupled transaction, the completion status is returned to the transaction master, which may then take some action. However, in a decoupled transaction, the master is not involved in the data acknowledgment at the destination bus and higher level protocols are required.

The error handling provided by the Universe is described for both coupled and decoupled transactions below.

2.7.1 Coupled Cycles

During coupled cycles, the Universe provides immediate indication of an errored cycle to the originating bus. VMEbus to PCI transactions terminated with target abort or master abort are terminated on the VMEbus with BERR*. The R_TA or R_MA bits in the PCI_CSR register (Table A-6) are set when it receives a target abort or master abort. For PCI to VMEbus transactions, a VMEbus BERR* received by the Universe is communicated to the PCI master as a target abort and the S_TA bit is set (Table A-6). No information is logged in either direction nor is an interrupt generated.

2.7.2 Decoupled Transactions

2.7.2.1 Posted Writes

The Universe provides the option of performing posted writes in both the PCI Slave Channel and the VME Slave Channel. Once data is written into the RXFIFO or TXFIFO by the initiating master (VMEbus or PCI bus respectively), the Universe provides positive acknowledgment of the cycle's completion. When the data in the FIFO is written to the destination slave by the Universe, it may subsequently receive a bus error instead of a positive acknowledgment. The Universe handles this situation by logging the errored transactions in one of two error logs and generating an interrupt. Each error log (one for VMEbus errors and one for PCI bus errors) is comprised of two registers: one for address and one for command or address space logging.

If the error occurs during a posted write to the VMEbus, the Universe uses the V_AMERR register (Table A-83) 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 address of the errored transaction is latched in the V_AERR register (Table A-84). An interrupt is generated on the VMEbus and / or PCI bus depending upon whether the VERR and VERR interrupts are enabled (see "The Universe Interrupt Channel" on page 2-58). The remaining entries of the offending transaction are purged.

If the error occurs during a posted write to the PCI bus, the Universe uses the L_CMDERR register (Table A-34) to log the command information for the transaction (CMDERR [3:0]). The address of the errored transaction is latched in the L_AERR register (Table A-35). An interrupt is generated on the VMEbus and / or PCI bus depending upon whether the VERR and LERR interrupts are enabled (see "The Universe Interrupt Channel" on page 2-58).

Under either of the above conditions (VME to PCI, or PCI to VME), the address that is stored in the log represents the most recent address the Universe generated before the bus error was encountered. For single cycle transactions, the address represents the address for the actual errored transaction. However, for multi-data beat transactions (block transfers on the VMEbus or burst transactions on the PCI bus) the log only indicates that an error occurred somewhere after the latched address. For a VMEbus block transfer, the logged address will represent the start of the block transfer. Since during PCI Slave Channel activity the Universe generates block transfers no larger than that programmed in the PABS field in the MAST_CTL register, the error will have occurred from the logged address up to the next aligned address. Similarly for a PCI burst transaction, the error will have occurred anywhere from the logged address up to the next burst aligned address.

In the case of PCI-initiated transactions, all data from the errored address up to the end of the initiating transaction is flushed from the TXFIFO. Since the Universe breaks PCI transactions at burst aligned addresses, the data is not flushed past this point. If the PCI master is generating bursts shorter than the Universe's aligned burst value (as set by the PABS in MAST_CTL register), then only data up to the end of that transaction is flushed).

Similarly in a posted write from the VMEbus, all data subsequent to the error in the transaction is flushed from the RXFIFO. However, the length of a VMEbus transaction differs from the length of the errored PCI bus transaction. For non-block transfers, the length always corresponds to one so only the errored data beat is flushed. However, if an error occurs on the PCI bus during a transaction initiated by a VMEbus block transfer, all data subsequent to the errored data beat in the block transfer is flushed from the RXFIFO. In the case of BLTs, this implies that potentially all data up to the next 256-byte boundary may be flushed. For MBLTs, all data up to the next 2-KByte boundary may be flushed.

Once an error is captured in a log, that set of registers is frozen against further errors until the error is acknowledged. The log is acknowledged and made available to latch another error by clearing the corresponding status bit in the VINT_STAT or LINT_STAT registers. Should a second error occur before the CPU has the opportunity to acknowledge the first error, another bit in the logs is set to indicate this situation (M_ERR bit).

Due to BERR* errata associated with PCI slave image posted writes, the VMIVME-7589 interface option provides auxiliary BERR handler logic. Chapter 3 discusses auxiliary functions such as the BERR interrupt, while errata are addressed in Chapter 5.

2.7.2.2 Prefetched Reads

In response to a block read from the VMEbus, the Universe initiates prefetching on the PCI bus (if the VMEbus slave image is programmed with this option). The transaction generated on the PCI bus is an aligned memory read transaction with multiple data beats extending to the aligned burst boundary (as programmed by PABS in the MAST_CTL register). Once an acknowledgment is given for the first data beat, an acknowledgment is sent to the VMEbus initiator by the assertion of DTACK*. Therefore, the first data beat of a prefetched read is coupled while all subsequent reads in the transaction are decoupled.

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 in this data beat is translated as a BERR* signal on the VMEbus. 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 is important to recognize that the target may be required to provide two target aborts before it is passed back to the VMEbus. Similarly, a target abort may never be passed back to the VMEbus if the block transfer does not extend to the point at which the error occurred.

2.7.2.3 DMA Errors

If bus errors occur during part of a DMA initiated transaction, the DMA halts, generates an interrupt (if the interrupt is enabled), and stops the address and count registers to give an indication of where the error occurred.

What the DMA does next depends upon whether the error occurs on the source bus (during the read) or on the destination bus (during the write). If the bus error occurs on the source bus, all DMA operations on that bus terminate immediately but any data previously queued within the DMAFIFO is still written to the destination bus. Once the DMAFIFO empties, the DMA generates an interrupt (as enabled by INT_LERR and INT_VERR in the DMA_GCS register, Table A-41). The bus on which the interrupt appears is determined by the DMA bits in the LINT_EN and VINT_EN registers (Table A-43 and Table A-47). The VERR (for VMEbus error) and LERR (for PCI bus error) bits in the DMA GCS register indicate which bus the error occurred on.

If the error occurs on the destination bus, the DMA immediately halts any operations on that bus, but continues reading from the source bus until the DMAFIFO no longer has room for another transaction. After the DMA stops on the source bus (or all data relevant to the command packet is read), the error bit in the DMA_GCSR register is set and an interrupt asserted (if enabled).

The address that the error occurred at will be indicated by the addresses stored in the VME and PCI Address registers. After an error on the source bus, the source bus registers will contain the address that the error occurred at. This will be independent of the size or type of transfers being performed on the bus. After an error on the destination bus, the destination address registers indicate a 128-bit window where the error occurred. The errored address will lie at an aligned 64-bit address somewhere before the stored address, or at an aligned 64-bit address contained in the address register. For example, if an error occurs on the destination bus and the address in the destination registers reads as 0x1000_1004, then the error can be said to lie somewhere in the range of 0x1000_0FF8 to 0x1000_1007.

Independent of the error occurring on the destination or source bus, the DMA_CTL register contains the attributes relevant to the particular DMA packet. It contains such information as: direction of transfer, address spaces accessed and data width. In addition, the DMA_TBC register provides the number of bytes remaining to transfer on the PCI side.

To generate an interrupt from a DMA error, there are two bits in the DGCS register (and one bit apiece in the VINT_EN and LINT_EN registers). In the DGCS register the INT_LERR bit enables the DMA to generate an interrupt to the Interrupt Channel encountering an error on the PCI bus. The INT_VERR enables the DMA to generate an interrupt upon encountering an error on the VMEbus. Upon reaching the Interrupt Channel, all DMA interrupts can be routed to either the PCI bus or VME bus by setting the appropriate bit in the enable registers. All DMA sources of interrupts (Done, Stopped, Halted, VME Error, and PCI Error) constitute a single interrupt into the Interrupt Channel.

2.7.2.4 Parity Errors

The Universe both monitors and generates parity information on the PAR# signal. The Universe monitors PAR# when it accepts data as a master during a read or a slave during a write. The Universe drives PAR# when it provides data as a slave during a read or a master during a write. The Universe also drives PAR# during the address phase of a transaction when it is a master and monitors PAR# during an address phase when it is the PCI slave. In both address and data phases, the PAR# signal provides even parity for C/BE#[3:0] and AD[31:0]. If the Universe is powered up in a 64-bit PCI environment, then PAR64 provides even parity for C/BE#[7:4] and AD[63:32].

The PERESP and SERR_EN bits in the PCI_CS register (Table A-6) determine whether or not the Universe responds to parity errors. Data parity errors are reported through the assertion of PERR# if the PERESP bit is set. Address parity errors, reported through the SERR# signal, are reported if both PERESP and SERR_EN are set. Regardless of the setting of these two bits, the D_PE (Detected Parity Error) bit in the PCI_CS register is set if the Universe encounters a parity error as a master or as a slave. The DP_D (Data Parity Detected) bit in the same register is only set if parity checking is enabled through the PERESP bit and the Universe detects a parity error while it is PCI master (i.e. it asserts PERR# during a read transaction or receives PERR# during a write).

No interrupts are generated by the Universe either as a master or as a slave in response to parity errors reported during a transaction. Parity errors are reported by the Universe through assertion of PERR# and by setting the appropriate bits in the PCI_CS register. If PERR# is asserted to the Universe while it is PCI master, the only action it takes is to set the DP_D. Regardless of whether the Universe is the master or slave of the transaction, and regardless which agent asserted PERR#, the Universe does not take any action other than to set bits in the PCI_CS register. The Universe continues with a transaction independent of any parity errors reported during the transaction.

Similarly, address parity errors are reported by the Universe (if the SERR_EN bit is set) by asserting the SERR# signal and setting the S_SERR (Signalled SERR#) bit in the PCI_CS register. Assertion of SERR# can be disabled by clearing the SERR_EN bit in the PCI_CS register. No interrupt is generated, and regardless of whether assertion of SERR# is enabled or not, the Universe does not respond to the access with DEVSEL#. Typically the master of the transaction times out with a master-abort. As a master, the Universe does not monitor SERR#. It is expected that a central resource on the PCI bus will monitor SERR# and take appropriate action.

2.8 Interrupter

The VMIVME-7589 interface option has two distinct sources of PCI interrupts: the first includes the Universe chip with its various PCI interrupt sources, while the second includes miscellaneous auxiliary logic, such as the mailbox registers and the BERR interrupt logic.

VMIC

The LINT#2, LINT#3, LINT#, LINT#5, LINT#6, and LINT#7 are not supported by the VMIVME-7589 interface option and should not be used. Refer to the following Universe chip specific material for a detailed description of PCI interrupt handling.

WARNING

Do not map any VMEbus interrupt to LINT#(2-7). Programming registers LINT_MAP0 or LINT_MAP1 with mappings to LINT#(2) - LINT#(7) will cause damage to the VMIVME-7589.

2.8.1 The Universe 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 2-11 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.

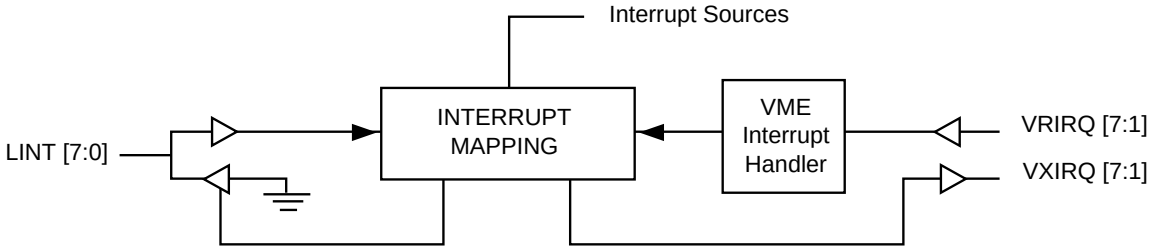


Figure 2-11 Interrupt Sources and Destinations

2.8.2 Universe 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 (see “Exclusive Accesses” on page 2-41)
  - -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 (Table A-43) and mapped to a single LINT signal through the LINT_MAP0 and LINT_MAP1 registers (Table A-45 and Table A-46). 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 (Table A-44). See Table 2-14 below for a list of the enable, mapping and status bits for PCI interrupt sources.

Table 2-13 PCI Interrupt Source Enabling, Mapping, and Status Bits

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

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 2-66).

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.

VMIC

LINT# 0 and LINT# 1 are supported. LINT# 0 maps to PCI INTA#. LINT# 1 maps to PCI SERR#.

2.8.3 Mailbox Interrupt Generation

VMIC

VMIC PCI-to-VMEbus interface option contains 4 separate 1-bit mailbox registers which provide software interrupt capability to the CPU via PCI INTA#. However, in order to access the registers, the user application must first setup and enable the PCI and VMEbus interfaces to the mailbox registers as follows:

1. First, setup the mailbox PCI base address by allocating 4K bytes of memory space and programming the PLX9060ES configuration space register offset 0x30 with this base address.
2. Next, setup the VME slave channel 0 such that it maps into the mailbox PCI window (i.e. translation offset maps to PCI mailbox base address). The mailbox window is a 4K PCI window, so the VME slave interface should be programmed to a 4K window. The application should use VME slave channel 0 which has a minimum window size of 4K.
3. Next, enable the mailbox window by setting bit 1 of the PLX9060ES configuration space register offset 0x30 via a PCI config write BIOS call.

Do not overwrite the base address; use Read-Modify-Write (RMW).

4. Finally, enable the mailbox interrupts by setting the appropriate mask bit in the system COMM register located at the SYS_BASE offset 0x0. Also, the user must set the PCI interrupt enable bit in the PLX9060ES Interrupt Control / Status Register located at the PLX_BASE offset 0x68.

Once the interface to the mailbox registers is enabled (i.e. both VME and PCI), a VMEbus master can generate a PCI INTA# interrupt by setting the Last Significant Bit (LSB) of the mailbox register.

The interrupt handler, in turn, should clear the interrupt source by clearing the LSB of the particular mailbox register. Since there is no hardware prioritizing of mailbox interrupts, the handler needs to poll the appropriate mailbox registers for active interrupts and process them as desired, clearing each source as it is processed.

2.8.4 Auxiliary BERR Interrupt Generation

The VMIVME-7589 interface option has an external BERR* circuit which, when enabled, is capable of generating a BERR* interrupt to the CPU via PCI INTA# or SERR#. This circuit only functions when the VMIVME-7589 interface option is VMEbus master. First, the circuit masks BERR* to the Universe chip and supplies DTACK* instead. This essentially disables the BERR circuitry inside the Universe chip and was incorporated to avoid a Universe chip errata associated with decoupled mode that might lock the VMEbus.

In addition, the circuit captures the address and address modifier code of the VMEbus BERR* cycle and sets the BERR status bit in the BERR status register located at the SYS_BASE offset 0x04. The BERR status bit will in turn generate a PCI interrupt, if enabled. The BERR interrupt can be mapped to PCI INTA# or PCI SERR# via the BERRIM bit in the COMM system register located at the SYS_BASE offset 0x00. Once a BERR cycle has been captured, it will remain captured until the BERR status bit has been processed by writing a 1 to the BERR status bit. The BERR captured address is available in the BERR address log register located at the SYS_BASE offset 0x0C. The BERR address modifier code is available in the BERR status register.

The BERR interrupt is enabled by setting the BERR_M bit in the BERR interrupt mask register located at the SYS_BASE offset 0x08, as well as setting the PCI local interrupt enable bit in the PLX9060ES interrupt control / status register located at the PLX_BASE offset 0x68.

2.8.5 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 (Table A-47) and mapped to a particular VME Interrupt level using the VINT_MAP0 and VINT_MAP1 registers (Table A-49 and Table A-50). 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 (Table A-48), 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 (Table A-51). See Table 2-14 below for a list of the enable, mapping and status bits for VMEbus interrupt sources.

Table 2-14 VMEbus Interrupt Source Enabling, Mapping, and Status Bits

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

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.

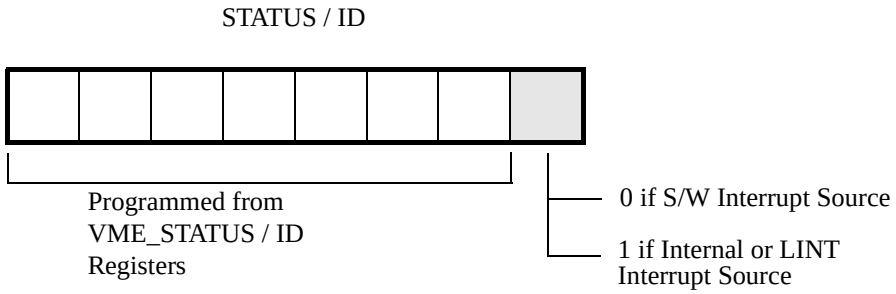


Figure 2-12 STATUS/ID Provided by Universe

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.

2.9 Interrupt Handling

2.9.1 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-47), 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-48).

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.

Since all the interrupt sources share a common PCI interrupt, the interrupt handler needs to poll all enabled sources and process them as required by the application. The Universe based interrupt sources are mapped to the PCI bus by programming the LINT_MAPI/0 registers as follows in Table 2-15.

Table 2-15 PCI bus LINT_MAP Registers

Interrupt Source Register Mapping	Corresponding PCI Interrupt
LINT#0	PCI INTA#
LINT#1	PCI SERR#

The LINT#2, LINT#3, LINT#4, LINT#5, LINT#6, and LINT#7 are not supported by the VMIVME-7589 interface option and should not be used. Refer to the following Universe chip specific material for a detailed description of PCI interrupt handling.

WARNING

Do not map any VMEbus interrupt to LINT#(2-7). Programming registers LINT_MAP0 or LINT_MAP1 with mappings to LINT#(2) - LINT#(7) will cause damage to the VMIVME-7589.

The VMIVME-7589 interface option supports LINT[1:0].

2.9.2 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 (Table A-43).

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 (see "VMEbus Requester" on page 2-9). 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 (Table A-52 to Table A-58), 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-49). 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.

2.9.2.1 Bus Error During VME IACK Cycle

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 (see “Bus Error Handling” on page 2-53). If the error occurs during an IACK cycle, the IACK# bit is set in the V_AMERR register (Table A-83). 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-52 to Table A-58), and generates an interrupt on the appropriate LINT pin (as mapped in the LINT_MAP0 register, Table A-45). 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-48) will show that both VIRQx, and VERR are active.

2.9.3 Internal Interrupt Handling

The Universe's internal interrupts are routed from several processes in the device (see Figure 2-13 on page 2-68). 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 2-16 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).

2.9.3.1 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 (Table A-47) triggers one (and only one) interrupt on the VMEbus on the level programmed in the VINT_MAP1 register (Table A-50). 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

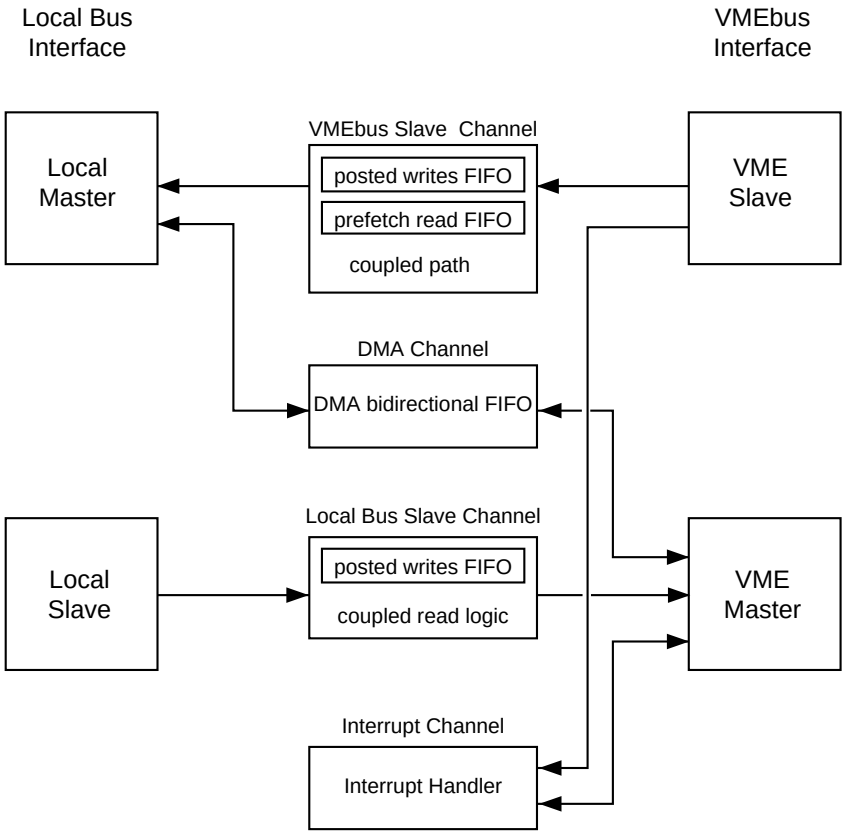


Figure 2-13 Universe Sources of Internal Interrupts

"one" to the SW_INT in the LINT_EN (Table A-43) register similarly 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.

Table 2-16 Internal Interrupt Routing

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	√	

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-44) 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.

2.9.3.2 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 (Table A-43) is set and the SW_IACK field in the LINT_MAP1 register (Table A-46) 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 (Table A-43) and mapped to a particular LINT pin using the LINT_MAP1 register (Table A-46). A status bit in the LINT_STAT register (Table A-44) indicates when the interrupt source is active, and is used to clear the interrupt once it has been serviced.

2.9.3.3 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 (Table A-59). This interrupt source can be used to indicate that ownership of the VMEbus is ensured during an exclusive access (see “Exclusive Accesses” on page 2-41). The interrupt is cleared by writing a one to the matching bit in the LINT_STAT register (Table A-44).

2.9.3.4 DMA Interrupt

The DMA module provides six possible interrupt sources:

- if the DMA is stopped (INT_STOP),
- if the DMA is halted (INT_HALT),
- if the DMA is done (INT_DONE),
- for PCI bus errors (INT_LERR),
- for VMEbus errors (INT_VERR), or
- if there is a PCI protocol error or if the Universe is not enabled as PCI master (INT_M_ERR).

All of these interrupt sources are ORed to a single DMA interrupt output line. When an interrupt comes from the DMA module, software must read the DMA status bits (Table A-41) to discover the originating interrupt source. The DMA interrupt can be mapped to either the VMEbus or one of the PCI interrupt output lines.

2.9.3.5 PCI and VMEbus Error Interrupts

Interrupts from bus errors (either PCI or VMEbus) are generated only when bus errors arise during decoupled writes. The bus error interrupt (from either a PCI or VMEbus error) can be mapped to either a VMEbus or PCI interrupt output line.

2.10 DMA Controller

The Universe uses a DMA controller for high performance data transfer between the PCI and VME buses. It is operated through a series of registers that control the source and destination for the data, length of the transfer and the transfer protocol to be used. There are two modes of operation for the DMA: Direct Mode, and Linked List Mode. In direct mode, the DMA registers are programmed directly by the software. In linked list mode, the registers are automatically loaded from PCI memory by the Universe, and the transfer described by these registers is executed. The block of DMA registers stored in PCI memory is called a command packet. A command packet may be linked to another command packet, such that when the DMA has completed the operations described by one command packet, it automatically moves on to the next command packed in the linked-list of command packets.

This section is broken into six major sub-sections:

1. “DMA Registers Outline” on page 2-71 describes in detail how the DMA is programmed from a register perspective.
2. “Direct Mode Operation” on page 2-77 describes how to operate the DMA when directly programming the DMA registers.
3. “Linked-List Operation” on page 2-80 describes how to operate the DMA when a linked-list of command packets describing DMA transfers is stored in PCI memory.
4. “FIFO Operation and Bus Ownership” on page 2-86 describes internally how the DMA makes use of its FIFO and how this affects ownership of the VME and PCI buses.
5. “DMA Interrupts” on page 2-90 describes the interrupts generated by the DMA
6. “DMA Error Handling” on page 2-90 describes how to handle errors encountered by the DMA

2.10.1 DMA Registers Outline

The DMA registers reside in a block starting at offset 0x200. Loosely speaking, they describe a single DMA transfer: where to transfer data from; where to transfer data to; how much data to transfer; and what data mode on the PCI and VME buses to use to transfer the data. A final register contains status and control information for the transfer. While the DMA is active, the registers are locked against any changes so that any writes to the registers will have no impact.

In direct-mode operation, these registers would be programmed directly by the user. In linked-list operation, they are repeatedly loaded by the Universe from command packets residing in PCI memory until the end of the linked-list is reached (see “Linked-List Operation” on page 2-80).

2.10.1.1 Source and Destination Addresses

The source and destination addresses for the DMA reside in two registers: the DMA PCI Bus Address Register (DLA register, Table A-38), and the DMA VMEbus Address Register (DVA register in Table A-39). The determination of which is the source address, and which is the destination is made by the L2V bit in the DCTL register (Table A-36). When set, the DMA transfers data from the PCI to the VMEbus. Hence DLA becomes the PCI source register and DVA becomes the VME destination register. When cleared, the DMA transfers data from the VME to PCI bus and DLA becomes the PCI destination register; DVA becomes the VME source register.

The PCI address may be programmed to any byte address in PCI memory space. It can not transfer to or from PCI IO or Configuration spaces.

The VME address may also be programmed to any byte address, and can access any VME address space from A16 to A32 in supervisory or non-privileged space, and data or program space. The setting of address space, A16, A24 or A32, is programmed in the VAS field of the DCTL register (Table A-36). The sub-spaces are programmed in the PGM and SUPER fields of the same register.

Note

Although the PCI and VME addresses may be programmed to any byte aligned address, they must be 8-byte aligned to each other (i.e. the low three bits of each must be identical). If not programmed with aligned source and destination addresses and the DMA is started, the DMA will immediately stop, set the protocol error bit (P_ERR) in the DCSR register (Table A-41), and if enabled to, generate an interrupt. Linked-list operations will cease.

The source and destination address registers are updated continuously by the DMA to allow the user to monitor DMA progress. If read during DMA activity, the DLA will return the address of the next PCI transfer, and the DTBC will return the number of bytes remaining to transfer on the PCI side. The DVA register may be incorrect from the actual active VME address by the contents of the FIFO. All of the DMA registers are locked against any changes by the user while the DMA is active. When stopped due to an error situation, these registers give an indication of where the error was encountered. See “DMA Error Handling” on page 2-90 for details. At the end of a transfer, these the DVA and DLA registers will point to the next address at the end of the transfer block, and the DTBC register will be zero.

2.10.1.2 Transfer Size

The DMA may be programmed through the DMA Transfer Byte Count register (DTBC register in Table A-37) to transfer any number of bytes from 1 byte to 16 MBytes. There are no alignment requirements to the source or destination addresses. Should the width of the data turnovers (8- through 64-bit on VME and 32- or 64-bit on PCI) not align to the length of the transfer or the source / destination addresses, the DMA will insert transfers of smaller width on the appropriate bus. For example, if a 15-byte transfer is programmed to start at address 0x1000 on the VME bus, and the width is set for D32, the DMA will perform three D32 transfers, followed by a D16 transfer, followed by a D08 transfer. The Universe does not generate unaligned transfers. On a 32-bit PCI bus, if the start address was 0x2000, the DMA would generate three data beats with all byte lanes enabled, and a fourth with three byte lanes enabled.

The DTBC register is not updated while the DMA is active (indicated by the ACT bit in the DGCS register). At the end of a transfer it will contain zero. If stopped by the user (via the STOP bit in the DGCS register) or the DMA encounters an error, the DTBC register contains the number of bytes remaining to transfer on the source side. See “DMA Error Handling” on page 2-90.

Starting the DMA while DTBC=0, will result in one of two situations. If the CHAIN bit in the DGCS register (Table A-41) is not set, the DMA will not start; it will perform no action. If the CHAIN bit is set, then the DMA loads the DMA registers with the contents of the command packet pointed to by the DCPP register (Table A-40), and starts the transfers described by that packet.

2.10.1.3 Transfer Data Width

The VME bus and PCI bus data widths are determined by three fields in the DCTL register (Table A-36). These fields affect the speed of the transfer. They should be set for the maximum allowable width that the targeted destination device is capable of accepting.

On the VMEbus, the DMA supports the following data widths:

- D08(EO), D16, D32, D64, D32BLT, and D64BLT (MBLT).

The DMA does not support D08(E), D08(O), and D16BLT. The width of the transfer is set with the VDW field in the DCTL register. Block transfers are enabled by setting the VCT bit in the same register.

The DMA may generate illegal VME cycles if programmed for D08 or D16 and block transfers are enabled.

The Universe may perform data transfers smaller than that programmed in the VDW field in order to bring itself into alignment with the programmed width. For example if the width is set for D32 and the starting VME address is 0x101, the DMA will perform a D08 cycle followed by a D16 cycle. Only once it has achieved the alignment set in the VDW field does it start D32 transfers. At the end of the transfer, the DMA will also have to perform more low-width transfers if the last address is not aligned to VDW. Similarly, if the VCT bit is set to enable block transfers, the DMA may perform non-block transfers to bring itself into alignment.

On the PCI bus, the DMA provides the option of performing 32- or 64-bit PCI transactions through the LD64EN bit in the DCTL register (Table A-36). If the Universe has powered-up on a 32-bit bus (see "Power-up Option Descriptions" on page 2-103), this bit will have no effect. If powered-up on a 64-bit bus, this bit can provide some performance improvements when accessing 32-bit targets on that bus. Following the PCI specification, before a 64-bit PCI initiator starts a 64-bit transaction, it engages in a protocol with the intended target to determine if it is 64-bit capable. This protocol typically consumes one clock period. To save bandwidth, the LD64EN bit can be cleared to bypass this protocol when it is known that the target is only 32-bit capable

2.10.1.4 DMA Command Packet Pointer

The DMA Command Packet Pointer (DCPP in Table A-40) points to a 64-bit aligned address location in PCI memory space that contains the next command packet to be loaded once the transfer currently programmed into the DMA registers has been successfully completed. When it has been completed (or the DTBC register is zero when the GO bit is set) the DMA reads the 32 byte command packet from PCI memory and executes the transfer it describes.

2.10.1.5 DMA Control and Status

The DMA General Control / Status Register (DGCS in Table A-41) contains a number of fields that control initiation and operation of the DMA as well as actions to be taken on completion.

DMA Initiation

Once all the parameters associated with the transfer have been programmed (source / destination addresses, transfer length and data widths, and if desired, linked lists enabled), the DMA transfer is started by setting the GO bit in the DGCS register. This causes to DMA to first examine the DTBC register. If it is non-zero, it latches the values programmed into the DCTL, DTBC, DLA, and DVA registers and initiates the transfer programmed into those registers. If DTBC=0, it checks the CHAIN bit in the DGCS register and if that bit is cleared it assumes the transfer to have completed and stops. Otherwise, if the CHAIN bit is set, it loads into the DMA registers the command packet pointed to by the DCPP register and initiates the transfer describe there.

If the GO bit is set, but the Universe has not been enabled as a PCI master with the BM (bus master enable) bit in the PCI_CSR register, or if the DVA and DLA contents are not 64-bit aligned to each other, the transfer does not start, a protocol error is indicated in the P_ERR bit in the DGCS register and, if enabled, an interrupt is generated.

If the DMA has been terminated for some reason (stopped, halted, or error), all DMA registers contain values indicating where the DMA terminated. Once all status bits have been cleared, the DMA may be restarted from where it left off by simply setting the GO bit. The GO bit will only have an effect if all status bits have been cleared. These bits include STOP, HALT, DONE, LERR, VERR, and P_ERR; all in the DGCS register. These bits are all cleared by writing "one" to them, either before or while setting the GO bit.

The GO bit always return a zero when read independent of what state the DMA is in. Clearing the bit has no impact at any time. The ACT bit in the DGCS register indicates whether the DMA is currently active. It is set by the DMA once the GO bit is set, and cleared when the DMA is idle. Generally, when the ACT bit is cleared, one of the other status bits in the DGCS register is set (DONE, STOP, HALT, LERR, VERR, or P_ERR), indicating why the DMA is no longer active.

DMA VMEbus Ownership

Two fields in the DGCS register determine how the DMA will share the VMEbus with the other two potential masters in the Universe (PCI slave channel, and Interrupt channel), and with other VME masters on the bus. These fields are:

- VON, which affects how much data the DMA will transfer before giving the opportunity to another master (either internal or external) to assume ownership of the bus.
- VOFF, which affects how long the DMA remains off the bus after the VON limit has been reached before the DMA will re-request it.

VON may be disabled by setting the field to zero. When set as such, the DMA will continue transferring data as long as it is able. With non-zero settings, the DMA will relinquish ownership of the bus when it reaches defined boundaries: 256, 512, 1K, 2k, 4K, 8K, or 16Kbytes. There are other conditions under which the DMA may relinquish bus ownership. See“FIFO Operation and Bus Ownership” on page 2-86 for details on the VMEbus request and release conditions for the DMA.

By setting VOFF to zero, the DMA will immediately re-request the bus once the VON boundary has been reached. Since the DMA operates in a round-robin fashion with the PCI slave channel, and a priority fashion with the interrupt channel, if either of these channels require ownership of the VMEbus, they will receive it at this time.

VOFF is only invoked when VMEbus tenure is relinquished due to encountering the VON boundary. When the bus is released due to other conditions (e.g., the DMA FIFO has gone full while reading from the VMEbus), it will be re-requested as soon as that condition is cleared. The VOFF timer can be programmed to various time intervals from 0µs to 1024µs. See “FIFO Operation and Bus Ownership” on page 2-86 for details on the VMEbus request and release conditions for the DMA.

DMA Completion and Termination

Normally, the DMA will continue processing its transfers and command packets until either it completes everything has been requested to, or it encounters an error. There are also two methods for the user to interrupt this process and cause the DMA to terminate prematurely: Stop and Halt. Stop causes the DMA to terminate immediately, while halt causes the DMA to terminate when it has completed processing the current command packet in a linked list.

When the STOP_REQ bit in the DGCS register is set by the user, it tells the DMA to cease its operations on the source bus immediately. Remaining data in the FIFO continues to be written to the destination bus until the FIFO is empty. Once the FIFO is empty, the STOP bit in the same register is set and, if enabled, an interrupt generated. The DMA register will contain the values that the DMA stopped at: the DTBC contains the number of bytes remaining in the transfer, the source and destination address registers contain the next address to be read / written, the DCPP contains the next command packet in the linked-list, and the DCTL contains the transfer attributes.

If read transactions are occurring on the VMEbus, then setting a stop request can be affected by the VOFF timer. If the STOP_REQ bit is set while the DMA is lying idle waiting for VOFF to expire before re-commencing reads, then the request remains pending until the VOFF timer has expired and the bus has been granted.

Halt provides a mechanism to interrupt the DMA at command packet boundaries during a linked-list transfer. In contrast, a stop requests the DMA to be interrupted immediately, while halt takes effect only when the current command packet is complete. A halt is requested of the DMA by setting the HALT_REQ bit in the DGCS register. This causes the DMA to complete the transfers defined by the current contents of the DMA registers and, if the CHAIN bit is set, load in the next command packet. The DMA then terminates, the HALT bit in the DGCS register is set, and, if enabled, an interrupt generated.

After a stop or halt, the DMA can be restarted from the point it left off by setting the GO bit; but before it can be re-started, the STOP bit and HALT bit must both be cleared.

Regardless of how the DMA stops whether normal, bus error or user interrupted, the DMA will indicate in the DGCS register why it stopped. The STOP and HALT bits get set in response to a stop or halt request. The DONE bit gets set when the DMA has successfully completed the DMA transfer, including all entries in the linked-list if operating in that mode. There are also three bits that are set in response to error

conditions: LERR in the case of target abort encountered on the PCI bus; VERR in the case of a bus error encountered on the VME bus; and P_ERR in the case that the DMA has not been properly programmed (the DMA was started with the BM bit in the PCI_CSR register not enabled, or the DLA and DVA registers were not 64-bit aligned, (see “Source and Destination Addresses” on page 2-72). Before the DMA can be restarted, each of these status bits must be cleared.

When the DMA terminates, an interrupt may be generated to VME or PCI buses. The user has control over which DMA termination conditions will cause the interrupt through the INT_STOP, INT_HALT, INT_DONE, INT_LERR, INT_VERR, and INT_P_ERR bits in the DGCS register.

2.10.2 Direct Mode Operation

When operated in direct mode, the Universe DMA is set through manual register programming. Once the transfer described by the DVA, DLA, DTBC and DCTL registers has been completed, the DMA sits idle awaiting the next manual programming of the registers.

Figure 2-14 describes the steps involved in operating the DMA in direct mode.

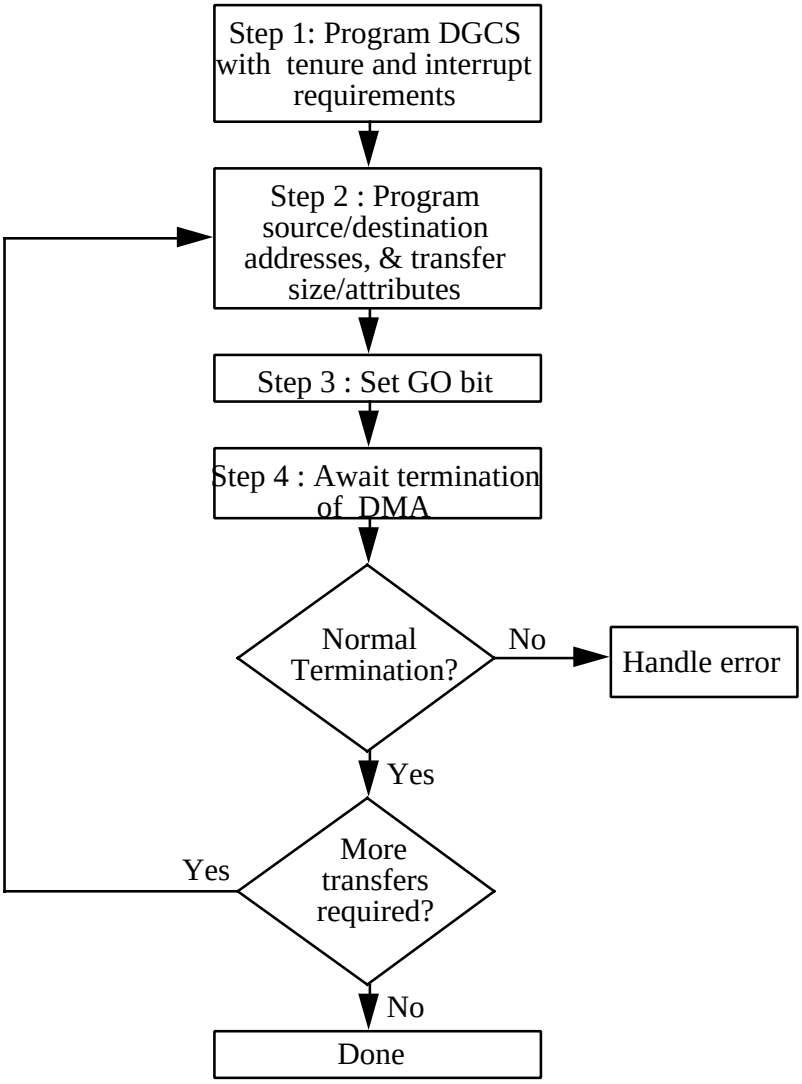


Figure 2-14 Direct Mode DMA transfers

In Step 1, the DGCS register is set up: the CHAIN bit is cleared, VON and VOFF are programmed with the appropriate values for controlling DMA VMEbus tenure, and the interrupt bits (INT_STOP, INT_HALT, INT_DONE, INT_LERR, INT_VERR, and INT_P_ERR) are programmed to enable generation of interrupts based on DMA termination events. DMA interrupt enable bits in the LINT_EN or VINT_EN bits should also be enabled as necessary (see “Universe PCI Interrupt Generation” on page 2-59 and “VMEbus Interrupt Generation” on page 2-63 for details on generating interrupts).

In Step 2, the actual transfer is programmed into the DMA: source and destination start addresses into the DLA and DVA registers, transfer count into the DTBC register, and transfer width, direction and VME address space into the DCTL register.

In Step 3, with the transfer programmed, the GO bit in DGCS must be set. Ensure that if any status bits (DONE, STOP, HALT, LERR, VERR, or P_ERR) remain set from a previous transfer they are cleared. This may be done during the same write cycle that sets the GO bit. The ACT bit will be set, and the DMA will then start transferring data, sharing ownership of the VME bus with the PCI slave and Interrupt channels and the PCI bus with the VME slave channel.

In Step 4, one waits for termination of the DMA transfers. The DMA will continue with the transfers until it:

- completes all transfers,
- is terminated early with the STOP_REQ bit, or
- encounters an error on the PCI or VME buses.

Each of these conditions will cause the ACT bit to clear, and a corresponding status bit to be set in the DGCS register. If enabled in step 1, an interrupt will also be generated. Once the software has set the GO bit, the software can monitor for DMA completion by either waiting for generation of an interrupt, or by polling the status bits. It is recommended that a background timer also be initiated to time-out the transfer. This will ensure that the DMA has not been hung up by a busy VMEbus, or other such system issues.

If an early termination is desired, perhaps because a higher priority operation is required, the STOP_REQ bit in the DGCS register can be set. This will stop all DMA operations on the source bus immediately, and set the STOP bit in the same register when the last piece of queued data in the DMA FIFO has been written to the destination bus. Attempting to terminate the transfer with the HALT_REQ bit will have no effect in direct mode operation since this bit only requests the DMA to stop between command packets in linked-list mode operation.

When the software has detected completion, it should check the status bits in the DGCS register to see the reason for completion. If one of the error bits have been set it proceeds into an error handling routine (see “DMA Error Handling” on page 2-90). If the STOP bit was set, the software should take whatever actions were desired when it set the STOP_REQ bit. For example, if it was stopped for a higher priority transfer, it might record the DLA, DVA and DTBC registers, and then reprogram them with the higher priority transfer. When that has completed it can restore the DVA, DLA and DTBC registers to complete the remaining transfers.

If the DONE bit was set, it indicates that the DMA completed its requested transfer successfully, and if more transfers are required, the software can proceed to step 2 to start a new transfer.

2.10.3 Linked-List Operation

Unlike direct mode, in which the DMA performs a single block of data at a time, linked-list mode allows the DMA to transfer a series of non-contiguous blocks of data without software intervention. Each entry in the linked-list is described by a command packet which parallels the DMA register layout. The data structure for each command packet is the same (see Figure 2-15 below), and contains all the necessary information to program the DMA address and control registers. It could be described in software as a record of eight 32-bit data elements. Four of the elements represent the four core registers required to define a DMA transfer: DCTL, DTBC, DVA, and DLA. A fifth element represents the DCPP register which points to the next command packet in the list. The least two significant bits of the DCPP element (the processed and null bits) provide status and control information for linked list processing.

The processed bit indicates whether a command packet has been processed or not. When the DMA processes the command packet and has successfully completed all transfers described by this packet, it sets the processed bit to “1” before reading in the next command packet in the list. This implies that the processed bit must be initially set for “0” by the user for it to be of use. This bit, when set to 1, indicates that this command packet has been disposed of by the DMA and its memory can be de-allocated or reused for another transfer description.

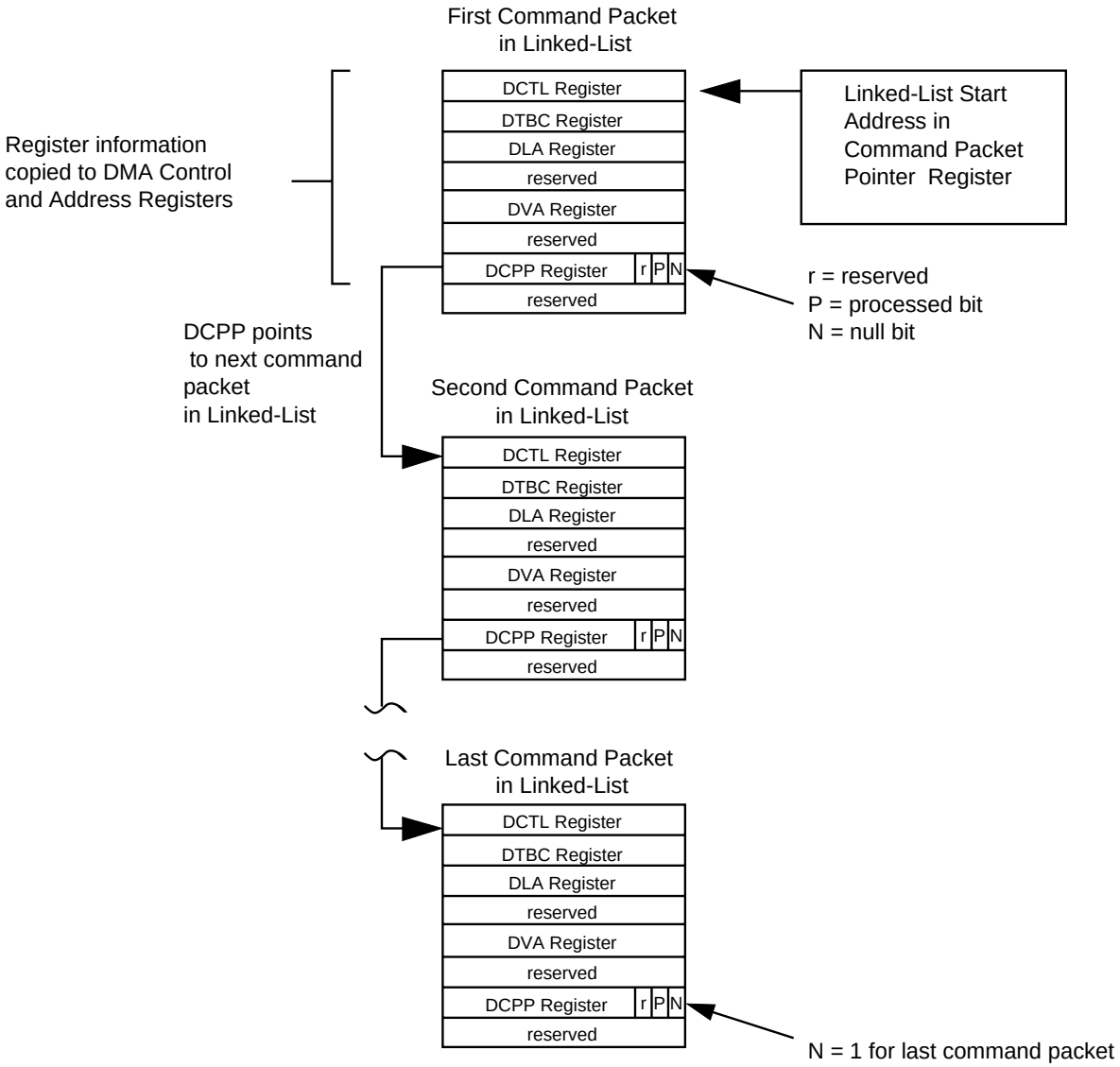


Figure 2-15 Command Packet Structure and Linked List Operation

The null bit indicates the termination of the entire linked list. If the null bit is set to “0”, the DMA processes the next command packet pointed to by the command packet pointer. If the null bit is set to “1” then the address in the command packet pointer is considered invalid and the DMA stops at the completion of the transfer described by the current command packet.

Figure 2-15 outlines the steps in programming the DMA for linked-list operation. In step 1, the DGCS register is set up: the CHAIN bit is set, VON and VOFF are programmed with the appropriate values for controlling DMA VMEbus tenure, and the interrupt bits (INT_STOP, INT_HALT, INT_DONE, INT_LERR, INT_VERR, and INT_P_ERR) are programmed to enable generation of interrupts based on DMA termination events. DMA interrupt enable bits in the LINT_EN or VINT_EN bits should also be enabled as necessary (“Universe PCI Interrupt Generation” on page 2-59 and “VMEbus Interrupt Generation” on page 2-63).

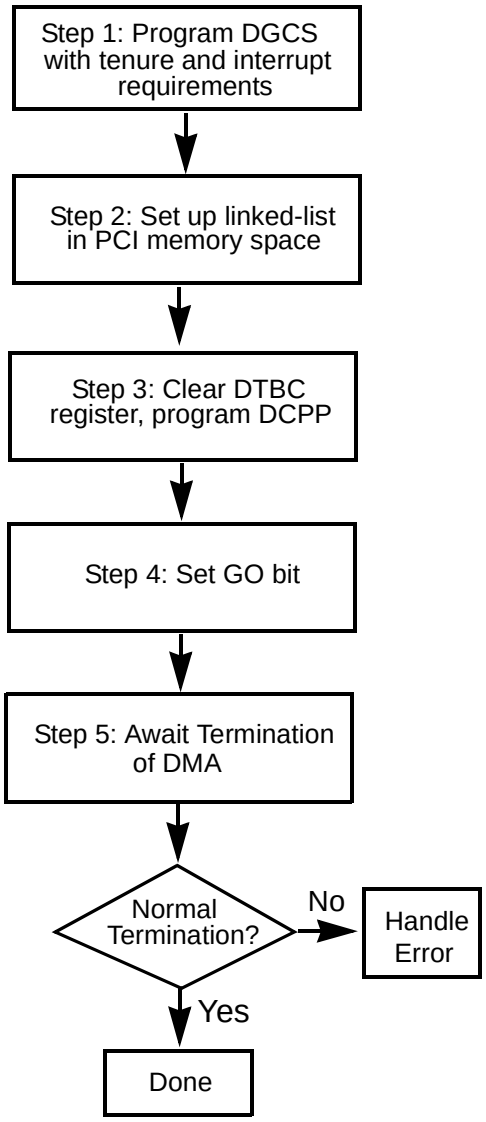


Figure 2-16 DMA Linked List Operation

Next the linked-list structure is programmed with the required transfers. The actual structure may be set up at any time with command packet pointers pre-programmed and then only the remaining DMA transfer elements need be programmed later. One common way is to set up the command packets as a circular queue: each packet points to the next in the list, and the last points to the first. This allows continuous programming of the packets without having to set-up or tear down packets later.

Once the structure for the linked-list is established, the individual packets are programmed with the appropriate source and destination addresses, transfer sizes and attributes.

To start the linked-list, clear the DTBC register, program the DCPP register to point to the first command packet in the list, and then set the GO bit in the DGCS register. If the DTBC register is non-zero when the GO bit is set, the DMA will first perform the transfers defined by the current contents of the DCTL, DTBC, DVA and DLA registers. Once that is complete it will then start the transfers defined by the linked-list pointed to in the DCPP register.

Once the DMA channel is enabled, it processes the first command packet as specified by the DCPP register. The DMA transfer registers are programmed by information in the command packets and the DMA transfer steps along each command packet in sequence (see Figure 2-15 on page 2-81). The DMA will terminate when it:

- processes a command packet with the null bit set indicating the last packet of the list
- is stopped with the STOP_REQ bit in the DGCS register
- is halted with the HALT_REQ bit in the DGCS register
- encounters an error on either the PCI or VME buses.

Each of these conditions will cause the ACT bit to clear, and a corresponding status bit to be set in the DGCS register. If enabled in step 1, an interrupt will also be generated. Once the software has set the GO bit, the software can monitor for DMA completion by either waiting for generation of an interrupt, by polling the status bits in the DGCS register, or by polling the "processed" bits of the command packets. It is recommended that a background timer also be initiated to time-out the transfer. This will ensure that the DMA has not been hung up by a busy VMEbus, or other such system issues.

Linked-list operation can be halted by setting the HALT_REQ bit in the DGCS register (Table A-41). When the HALT_REQ bit is set, the DMA terminates when all transfers defined by the current command packet is complete. It then loads the next command packet into its registers. The HALT bit in the DGCS register is asserted, and the ACT bit in the DGCS register is cleared. The DMA can be restarted by clearing the HALT status bit and setting the GO bit if desired during the same register write. If the DMA is indeed restarted, the ACT bit is set by the Universe and execution continues as if no HALT had occurred: i.e., the Universe processes the current command packet (see Figure 2-15 on page 2-81).

In contrast to a halt, the DMA can also be immediately terminated through the STOP_REQ bit. This will stop all DMA operations on the source bus immediately, and set the STOP bit in the same register when the last piece of queued data in the DMA FIFO has been written to the destination bus.

Once stopped, the DVA, DLA and DTBC registers contain values indicating the next addresses to read / write and the number of bytes remaining in the transfer. Clearing the STOP bit and setting the GO bit will cause the DMA to start up again from where it left off, including continuing with subsequent command packets in the list.

If the DMA is being stopped to insert a high priority DMA transfer, the remaining portion of the DMA transfer may be stored as a new command packet inserted at the top of the linked list. A new command packet with the attributes of the high priority transfer is then placed before that one in the list. Now the linked list is set up with the high priority packet first, followed be the remainder of the interrupted packet, followed in turn by the rest of the linked list. Finally, the DTBC register is cleared and the DCPP programmed with a pointer to the top of the list where the high priority command packet has been placed. When the GO bit is set (after clearing the STOP status bit in the DGCS register), the DMA will perform the transfers in the order set in the linked list. For more details on updating the linked list see the following section, “Linked List Updating”.

DMA transfers continue until the DMA encounters a command packet with the null bit set to “1”, indicating that the last packet has been reached. At this point, the DMA stops, the DONE bit is set, and the ACT flag is cleared. As it completes the transfers indicated by each command packet, the DMA sets the "processed" bit in that command packet before reading in the next command packet and processing its contents.

2.10.3.1 Linked List Updating

The Universe provides a mechanism which allows the linked list to be updated with additional linked list entries without halting or stopping the DMA. This takes place through the use of a semaphore in the device: the UPDATE bit in the D_LLUE register, Appendix A, Table A-42. This bit ensures that a command packet is not altered for the linked-list update process at the same time that the DMA is reading the command packet into the DMA registers.

Adding to a linked list begins by writing a “1” to the UPDATE bit . The DMA checks this bit before proceeding to the next command packet. If the UPDATE bit is “0”, then the DMA locks the UPDATE bit against writes and proceeds to the next command packet. If the UPDATE bit is “1”, then the DMA waits until the bit is cleared before proceeding to the next command packet. Therefore, setting the UPDATE bit is a means of stalling the DMA at command packet boundaries while local logic updates the linked list.

In order to ensure that the DMA is not currently reading a command packet during updates, the update logic must write a “1” to the UPDATE bit and read a value back. If a “0” is read back from the UPDATE bit, then the DMA is currently reading a command packet and has locked the UPDATE bit against writes. If a “1” is read back from the UPDATE bit, then the DMA is idle or processing a transaction and command packets can be updated. If the DMA attempts to proceed to the next command packet during the update, it will encounter the set UPDATE bit and wait until the bit is cleared.

If a set of linked command packets has already been created with empty packets at the end of new transfers, adding to the end of the current linked list takes the following procedure:

1. Get UPDATE valid (write “1”, read back “1”),
2. Program attributes for new transfer in next available packet in list.
3. Change “null” pointer (on previous tail of linked list),
4. Release update (clear the UPDATE bit).

After updating the linked list, the DMA controller will be in three possible conditions:

1. It may be active and working its way through the linked list. In this case, no further steps are required.
2. The DMA may be idle (done) because it reached the final command packet. If a full set of linked command packets had already been created ahead of time, then the DCPP register will point to the most recently programmed command packet, and the DTBC register will be zero. The DMA can be started on the new packet by simply clearing the DONE bit and setting the GO bit in the DGCS register. If a set of command packets had not been created ahead of time, the DCPP register may not be programmed to any valid packet, and will need programming to the newly programmed packet.
3. The DMA has encountered an error. In this circumstance, see “DMA Error Handling” on page 2-90 for how to handle DMA errors.

Operation may be considerably simplified by ensuring that sufficient command packets have been created during system initialization, probably in a circular queue. In this fashion, when a new entry is added to the list, it is simply a matter of programming the next available entry in the list with the new transfer attributes and changing the previously last packet's "null" bit to zero. The DCPP register will be guaranteed to point to a valid command packet, so upon updating the list, both cases 1 and 2 above can be covered by clearing the DONE bit and setting the GO bit. This will have no effect for case 1 since the DMA is still active, and will restart the DMA for case 2.

If an error has been encountered by the DMA (case 3) , setting the GO bit and clearing the DONE bit will not be sufficient to restart the DMA—the error bits in the DGCS register will also have to be cleared before operation can continue.

2.10.4 FIFO Operation and Bus Ownership

The DMA uses a 16-entry deep by 64-bit wide bi-directional FIFO to increase performance for DMA transfers. In general, the DMA reads data from the source, and stores it as transactions in the FIFO. On the destination side, the DMA requests ownership of the master and once granted begins transfers. Transfers stop on the source side when the FIFO fills, and on the destination side when the FIFO empties.

2.10.4.1 PCI to VME Transfers

On the PCI side, the PCI bus is requested for the current operation once the available space in the DMAFIFO is greater than or equal to the programmed PCI aligned burst size (PABS in the MAST_CTL register). The DMA channel fills the DMAFIFO using PCI read transactions with each transaction broken at address boundaries determined by the programmed PCI aligned burst size. This ensures that the DMA makes optimal use of the PCI bus by always generating bursts of 32 or 64 bytes with zero wait states.

The DMA packs read data into the DMAFIFO to the full 64-bit width of the FIFO, independent of the width of the PCI bus, or the data width of the ensuing VMEbus transaction. The PCI read transactions continue until either the DMA has completed the full programmed transfer, or there is insufficient room available in the DMAFIFO for a full transaction. The available space required for another burst read transaction is eight entries if PABS is programmed to 64 bytes, or four entries if programmed to 32 bytes. Since the VMEbus is typically much slower than the PCI bus, the DMAFIFO may fill frequently during PCI to VME transfers. When the DMAFIFO fills, the PCI bus is free for other transactions (for example, between other devices on the bus or possibly for use by the Universe's VME Slave Channel). The DMA only resumes read transactions on the PCI bus when the DMAFIFO has space for another aligned burst size transaction.

Once eight entries have been queued into the FIFO, the DMA requests ownership of the Universe's VME master interface (see "VMEbus Requester" on page 2-9 on how the VME Master Interface is shared between the DMA, the PCI slave channel, and the Interrupt channel). The Universe maintains ownership of the master interface until:

- the DMAFIFO is empty,
- the DMA block is complete,
- the DMA is stopped,
- a linked list is halted,
- the DMA encounters an error, or
- the DMA VME tenure limit (VON in the DGCS register).

The DMA can be programmed to limit its VMEbus tenure to fixed block sizes using the VON field in the DGCS register (Table A-41 on page -49). With VON enabled, the DMA will relinquish ownership of the master interface at defined address boundaries: 256 bytes, 512 bytes, 1 Kbyte, 2 Kbytes, 4 Kbytes, 8 Kbytes, or 16 Kbytes. By default, VON is disabled and the DMA VMEbus tenure will be limited by one of the other mechanisms listed in the previous paragraph.

To further control the DMA's VMEbus ownership, the VOFF timer in the DGCS register can be used to program the DMA to remain off the VMEbus for a specified period when VME tenure is relinquished. The VOFF timer can be set to 16 µs, 32 µs, 64 µs, 128 µs, 256 µs, 512 µs, or 1024 µs and only becomes active when the DMA reaches a boundary determined by the VON counter. The timer is disabled by default. This timer can ensure a predetermined amount of VMEbus bandwidth is available to other VME devices (including the PCI slave channel and the Interrupt channel) during DMA transfers. The VOFF timer is not invoked when the DMAFIFO empties of by any other mechanism other than expiry of the VON byte count.

The DMA Channel unpacks the 64-bit data queued in the DMAFIFO to whatever the programmed transfer width is on the VMEbus (e.g. D16, D32, or D64). The VME Master Interface delivers the data in the DMAFIFO according to the VMEbus cycle type programmed into the DCTL register in Appendix A, (Table A-36, and see "DMA Controller" on page 2-71). The DMA provides data to the VMEbus until:

- the DMAFIFO empties, or
- the DMA VMEbus Tenure Byte Count (VON in the DMA_GCSR register, Table A-41) expires.

If the DMAFIFO empties, transfers on the VMEbus stop and, if the cycle being generated is a block transfer, then the block is terminated (AS* negated) and VMEbus ownership is relinquished by the DMA. The DMA does not re-request VMEbus ownership until another eight entries are queued in the DMAFIFO, or the DMA Channel has completed the current Transfer Block on the PCI bus (see "VMEbus Release" on page 2-10).

PCI bus transactions are the full width of the PCI data bus with appropriate byte lanes enabled. The maximum VMEbus data width is programmable to 8, 16, 32, or 64 bits. Byte transfers can be only of type DO8 (EO). Because the PCI bus has a more flexible byte lane enabling scheme than the VMEbus, the Universe may be required to generate a variety of VMEbus transaction types to handle the byte resolution of the starting and ending addresses (see "Data Transfer" on page 2-34).

2.10.4.2 VME to PCI Transfers

With DMA transfers in this direction, the DMA Channel begins to queue data in the DMAFIFO as soon as there is room for eight entries (64 bytes) in the DMAFIFO. When enough room is available, the DMA will request the VMEbus (through the VME master interface) and begin reading data from the VMEbus. The Universe maintains VMEbus ownership until:

- the DMAFIFO is full,
- the DMA block is complete,
- the DMA is stopped,
- a linked list is halted,
- the DMA encounters an error, or
- the VMEbus VME tenure limit is reached (VON in the DGCS register).

The DMA can be programmed to limit its VMEbus tenure to fixed block sizes using the VON field in the DGCS register in Appendix A, Table A-41. With VON enabled, the DMA will relinquish ownership of the master interface at defined address boundaries: 256 bytes, 512 bytes, 1 Kbyte, 2 Kbytes, 4 Kbytes, 8 Kbytes, or 16 Kbytes. By default, VON is disabled and the DMA VMEbus tenure will be limited by one of the other mechanisms listed in the previous paragraph.

To further control the DMA's VMEbus ownership, the VOFF timer in the DGCS register can be used to program the DMA to remain off the VMEbus for a specified period when VME tenure is relinquished. The VOFF timer can be set to 16 µs, 32 µs, 64 µs, 128 µs, 256 µs, 512 µs, or 1024 µs and only becomes active when the DMA reaches a boundary determined by the VON counter. The timer is disabled by default. This timer can ensure a predetermined amount of VMEbus bandwidth is available to other VME devices (including the PCI slave channel and the Interrupt channel) during DMA transfers. The VOFF timer is not invoked when the DMAFIFO empties of by any other mechanism other than expiry of the VON byte count.

On the PCI side, entries in the DMAFIFO are delivered to the PCI bus as PCI write transactions as soon as the number of entries reaches the PCI aligned burst size (as programmed in the PABS bit in the MAST_CTL register). If the PCI bus responds too slowly, the DMAFIFO runs the risk of filling before write transactions can begin at the PCI master interface. Once the DMAFIFO reaches a "nearly full" state (corresponding to three entries remaining) then the DMA requests that the VME master interface complete its pending operations and stop. The pending read operations typically fill the DMAFIFO. Once the pending VMEbus reads are completed (or the VON timer expires), the DMA relinquishes VMEbus ownership and only re-requests the VME master interface once eight entries again become available in the DMAFIFO. If the bus was released due to encountering a VON boundary, the bus is not re-requested until the VOFF timer expires.

PCI bus transactions are the full width of the PCI data bus with appropriate byte lanes enabled. The maximum VMEbus data width is programmable to 8, 16, 32, or 64 bits. Byte transfers can be only of type DO8 (EO). Because the PCI bus has a more flexible byte lane enabling scheme than the VMEbus, the Universe may be required to generate a variety of VMEbus transaction types to handle the byte resolution of the starting and ending addresses (see "Universe as PCI Slave" on page 2-33).

2.10.5 DMA Interrupts

The interrupt channel in the Universe handles a single interrupt sourced from the DMA channel which it routes to either the VME or PCI buses via the DMA bits in the LINT_EN and VINT_EN registers. There are six internal DMA sources of interrupts and these are all routed to this single interrupt. Each of these six sources may be individually enabled, and are listed in Table 2-17 below. Setting the enable bit enables the corresponding interrupt source.

Table 2-17 DMA Interrupt Sources and Enable Bits

Interrupt Source	Enable Bit
Stop Request	INT_STOP
Halt Request	INT_HALT
DMA Completion	INT_DONE
PCI Bus Error	INT_LERR
VMEbus Error	INT_VERR
Protocol Error	INT_M_ERR

Once an enabled DMA interrupt has occurred, regardless of whether the LINT_EN or VINT_EN enable bits have been set, the corresponding DMA bit in the LINT_STAT and VINT_STAT registers are set. Each one must be cleared independently. Clearing one does not clear the other.

2.10.6 DMA Error Handling

While the DMA is operating normally, the ACT bit in the DGCS register will be set (Table A-41). Once the DMA has terminated, it will clear this bit, and set one of six status bits in the same register. The DONE bit will be set if the DMA completed all its programmed operations normally. If the user interrupted the DMA, either the STOP or HALT bits will be set. If an error has occurred, one of the remaining three bits, LERR, VERR, or P_ERR, will be set. All six forms of DMA terminations can be optionally set to generate a DMA interrupt by setting the appropriate enable bit in the DGCS register (see “DMA Interrupts” on page 2-90).

LERR is set if the DMA encounters an error on the PCI bus: either a Master Abort or Target Abort. Bits in the PCI_CSR register will indicate which of these conditions caused the error.

VERR is set if the DMA encounters a bus error on the VME bus. This will be exclusively through a detected assertion of BERR* during a DMA cycle.

P_ERR is asserted if the GO bit in the DGCS register is set to start the DMA, and the DMA has been improperly programmed either because the BM bit in the PCI_CSR has not been set to enable PCI bus mastership, or the source and destination start addresses are not aligned (see “Source and Destination Addresses” on page 2-72).

If the bus error occurs on the source bus, all DMA operations terminate immediately on the source bus and any data previously queued within the DMAFIFO is written to the destination bus. Once the DMAFIFO empties, the error status bit is set and the DMA generates an interrupt (as enabled by INT_LERR and INT_VERR in the DGCS register—see “DMA Interrupts” on page 2-90).

If the error occurs on the destination bus, the DMA immediately halts any operations on that bus, but continues reading from the source bus until the DMAFIFO no longer has room for another transaction. After the DMA stops on the source bus (or all data relevant to the command packet is read), the error bit in the DGCS register is set and an interrupt asserted (if enabled).

The address at which the error occurred will be indicated by the addresses stored in the VME and PCI Address registers (DVA and DLA). After an error on the source bus, the source bus address register will contain the address at which the error occurred. This will be independent of the size or type of transfers being performed on the bus. After an error on the destination bus, the destination bus address register will contain the address at which the error occurred.

Independent of the error occurring on the destination or source bus, the DMA_CTL register contains the attributes relevant to the particular DMA transaction. In addition, the DTBC register provides the number of bytes remaining to transfer on the PCI side.

After an error or DMA completion, the DMA can be restarted with GO. However, the DMA will re-start only after the various status bits are cleared (including the DONE bit). The status bits are cleared by writing 1 to them. This may be done before or while setting the Go bit.

2.11 Registers

A VMIVME-7589 interface option has two primary groups of registers:

- PCI configuration space registers
- User programmable registers

The interface option also has two sets of PCI configuration space registers; the first group is located in the Universe chip and the other groups located in the PLX9060ES PCI target interface chip. These registers are initialized by the PCI BIOS during boot time. The registers can be accessed by PCI BIOS calls or VMIC control function library software. The user will need to access these registers primarily to obtain the base addresses for the various user programmable registers. A complete description of the PCI configuration space registers is included in Appendix A in Figure A-1 and in Figure A-2.

The interface contains four groups of user programmable registers:

- Auxiliary Functions Global Interrupt Mask Register
- System Registers
- Mailbox Registers, and
- Universe Control and Status Registers (UCSR).

Table 2-18 shows the base address mapping for each of the register sets. These base addresses are mapped by the PCI BIOS at boot time.

Table 2-18 Interface Base Address Map

Register Group	Base Address Name	Size (Bytes)	Type	Config Space Location
Auxiliary Functions Global Interrupt Mask Register	**AFI_BASE**	**128K**	**Memory**	**$10 (PLX)**
System registers	SYS_BASE	4K	Memory	$18 (PLX)
Mailbox registers	MB_BASE	4K	Memory	$30 (PLX)
Universe registers (UCSR)	UNIV_BASE	64K (1)	Memory	$10 (Universe)
Notes: (1) Due to a Universe chip errata, 4K register maps into 64K space.				

The Auxiliary Functions Global Interrupt Mask Register enables the auxiliary functions logic to route its interrupts to the PCI bus via the PLX9060ES interface chip. This register is located at base address AFI_BASE offset 0x68. A complete description of the register is included in Appendix A, Table A-4.

The system registers consist of four registers located at base address SYS_BASE. These registers provide additional control/status not contained within the Universe chip, such as master/slave endian conversion and non-slot 1 bus time-out timer. A complete description of all System Registers is included in Appendix A, Table A-2, the System Register Map.

The mailbox registers consist of four1-bit registers located at base address MB_BASE**.** These registers provide interrupt capability to the CPU via PCI INTA#. A complete description of all Mailbox Registers is included in Appendix A, Table A-3, the Mailbox Register Map.

Most of the VMIVME-7589 interface option's programmable registers are located within the Universe chip.The Universe chip contains a 4K register set located at base address UNIV_BASE. The VMIVME-7589 interface option contains a configurable jumper which allows the Universe Based Registers to be located in memory space or I/O space. The Universe Register Map

Due to a Universe chip errata which maps the 4K register set into 64K space, the registers should be configured to be mapped into memory space. The address of the registers is contained in the Universe's PCI configuration space base address register 0 (i.e. configuration space offset 0x10).

The Universe Control and Status Registers (UCSR) occupy 4 Kbytes of internal memory. This 4 Kbytes is logically divided into three groups (see Figure 2-17 below):

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

The Universe registers are little-endian.

The access mechanisms for the UCSR are different depending upon whether the register space is accessed from the PCI bus or VMEbus. Register access from the PCI bus and VMEbus is discussed below.

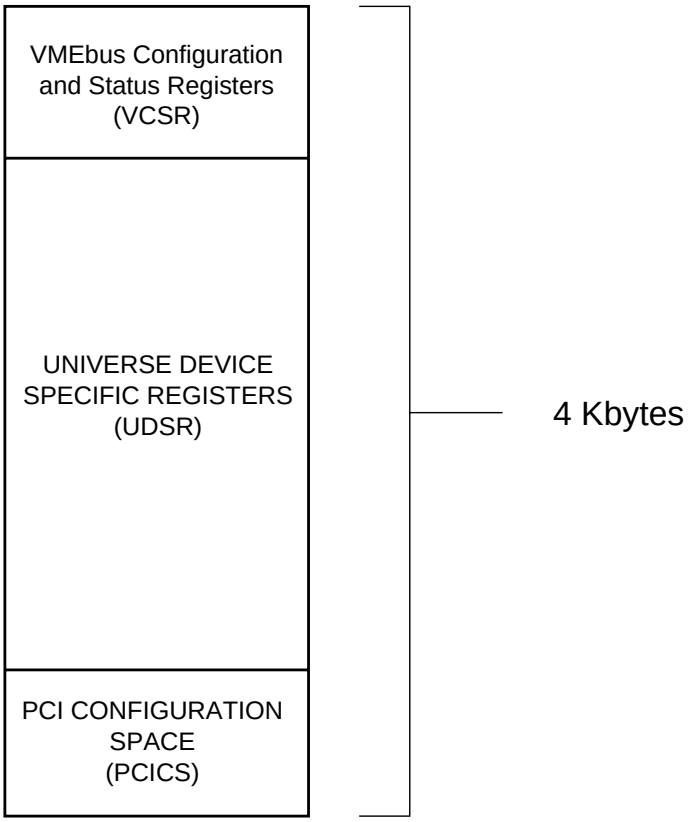


Figure 2-17 Universe Control and Status Register Space

2.11.1 Register Access from the PCI Bus

There are two mechanisms to access the UCSR space from the PCI bus: through Configuration space or through the PCI defined base address register (Reference Table A-9, “PCI Configuration Base Address Register (PCI_BS),” on page A-16 in Appendix A).

2.11.1.1 PCI Configuration Access

When the UCSR space is accessed as Configuration space, it means that the access is externally decoded and the Universe is notified via IDSEL# (much like a standard chip select signal). Since the register location is encoded by a 6-bit register number (a value used to index a 32-bit chunk of configuration space), only the lower 256 bytes of the UCSR can be accessed as configuration space (this corresponds to the PCICS in the UCSR space, see Figure 2-18 on page 2-96). Thus, only the PCI configuration registers are accessible through PCI Configuration cycles.

To access the other UCSR registers through Memory or I/O space, the base address of the PCI_BS register (Table A-9) must first be programmed, since in Memory or I/O space the UCSRs are located as address offsets from the base address programmed into the BS field in the PCI_BS register.

2.11.1.2 Memory or I/O Access

The UCSR space can be configured as Memory or I/O space at power-up, but not both (see “Power-Up Options” on page 2-102). There is an internal inversion on the power-up pin (VA[1]). When the VA[1] pin is sampled low at power-up, the device starts-up in I/O space and the SPACE bit of the PCI_BS register (Table A-9) is set to 1. When VA[1] is sampled high at power-up, the Universe starts-up in Memory space. The user is free to locate the registers anywhere in the 32-bit Memory or I/O space; however, the registers are not prefetchable. Once configured at power-up, the UCSR space can not be reconfigured until the next power-up sequence, although their location within that space may be reprogrammed at any time.

All of the UCSR space (4 Kbytes) is accessible as Memory or I/O space from the PCI bus (see Figure 2-18 below) once the PCI_BS register (Table A-9) has been programmed. The Base Address field of the PCI_BS register can be programmed through PCI Configuration space (as described above) or through a VME access.

Due to a Universe chip errata which maps the 4K register set into 64K space, the registers should be configured to be mapped into memory space. The address of the registers is contained in the Universe’s PCI configuration space base address register 0 (i.e. config space offset 0x10).

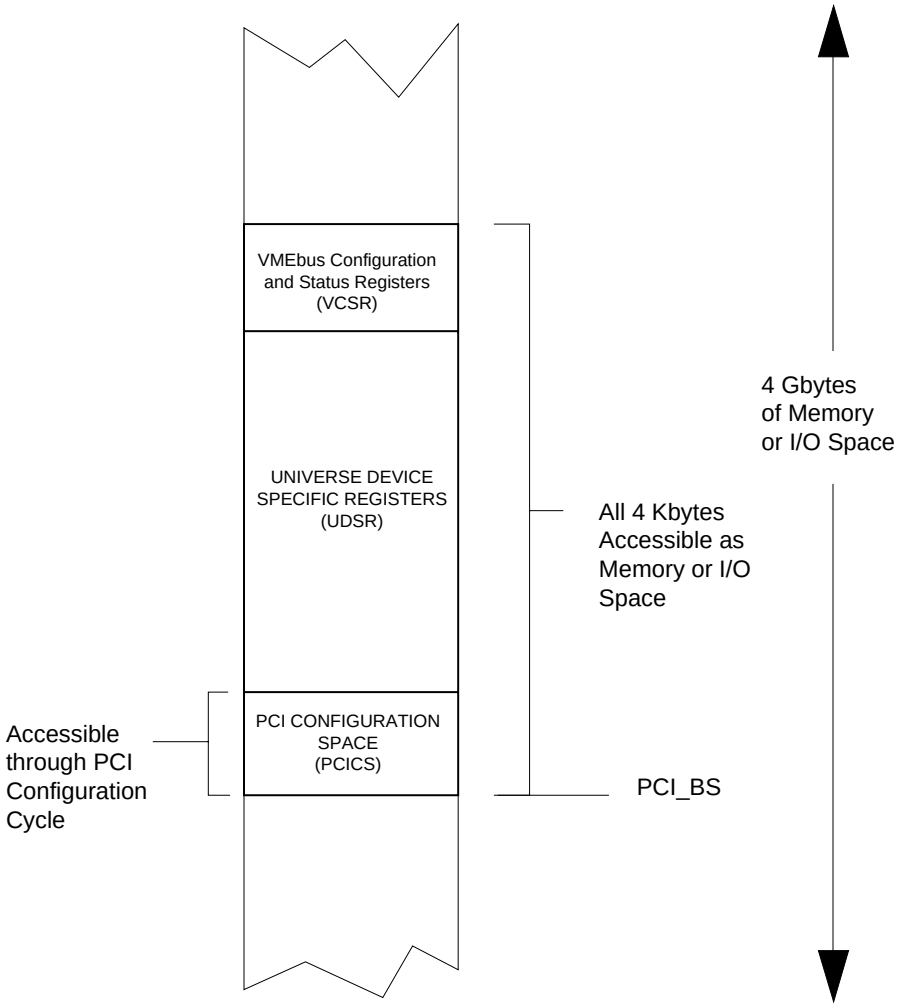


Figure 2-18 PCI Bus Access to UCSR as Memory or I/O Space

2.11.2 Register Access from the VMEbus

There are two mechanisms to access the UCSR space from the VMEbus. One method uses a VMEbus Register Access Image (VRAI) which allows the user to put the UCSR in an A16, A24 or A32 address space. The VRAI approach is useful in systems not implementing CR/CSR space as defined in the VME64 specification. The other way to access the UCSR is as CR/CSR space, where each slot in the VMEbus system is assigned 512 Kbytes of CR/CSR space.

Each method is discussed below.

2.11.2.1 VMEbus Register Access Image

The VMEbus register access image is defined by the following register fields:

Table 2-19 Programming the VMEbus Register Access Image

Field	Register Bits	Description
address space	VAS in Table A-79	one of A16, A24, A32
base address	BS[31:12] in Table A-80	lowest address in the 4Kbyte slave image
slave image enable	EN in Table A-79	enables VMEbus register access image
mode	SUPER in Table A-79	Supervisor and/or Non-Privileged
type	PGM in Table A-79	Program and/or Data

The VMEbus Register Access Image occupies 4 Kbytes in A16, A24 or A32 space (depending upon the programming of the address space described in Table 2-19 above, see Figure 2-19 below). All registers are accessed as address offsets from the VRAI base address programmed in the VRAI_BS register (Table A-80). The image can be enabled or disabled using the EN bit in the VRAI_CTL register (Table A-79).

Note that the VRAI base address can be configured as a power-up option (see “Power-up Option Descriptions” on page 2-103).

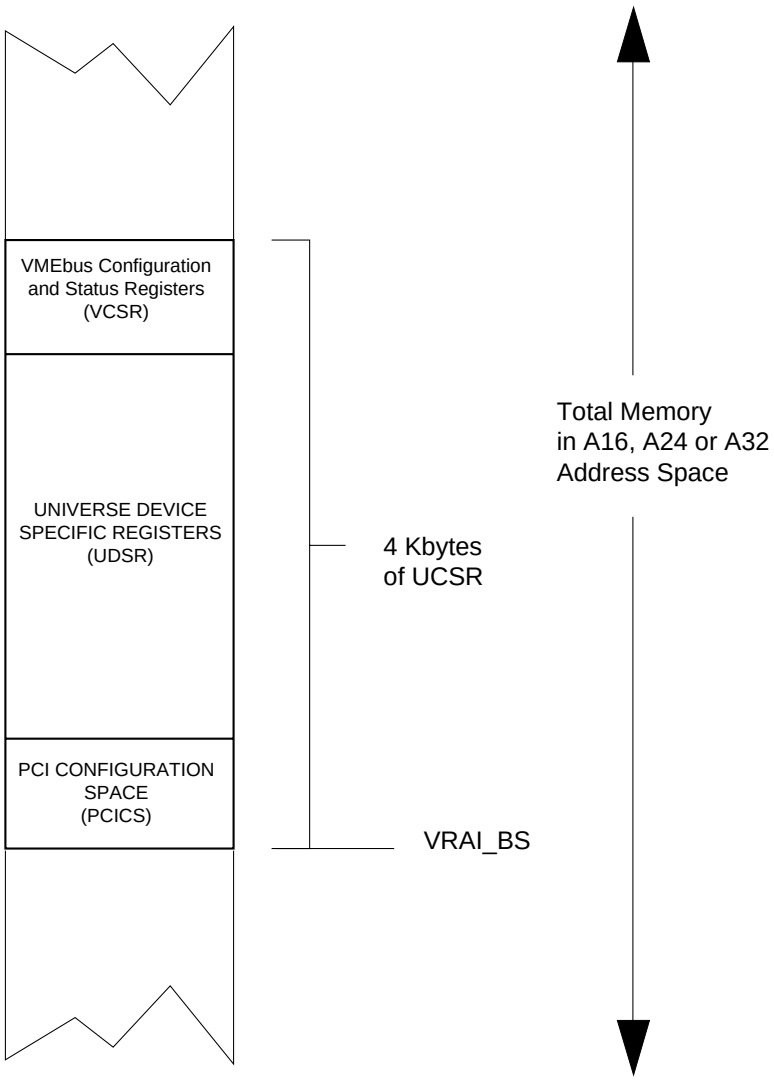


Figure 2-19 UCSR Access from the VMEbus Register Access Image

The interface option does not support CR/CSR space.

2.12 Utility Functions

2.12.1 Resets

This section is divided in three sections. The first section lists the pins and registers that are involved in the reset circuitry. The second section presents and explains the reset circuitry diagram. The third section provides important suggestions and warnings about configuring the Universe reset circuitry.

2.12.1.1 Overview of Reset Support

The Universe provides a number of pins and registers for reset support. Pin support is summarized in Table 2-20.

Table 2-20 Universe Signals Involved in Reset Circuitry

Interface and direction	Pin name	Long Name	Brief Description
JTAG Input	TRST#	JTAG Test Reset	Provides asynchronous initialization of the TAP controller in the Universe.
VME Input	VRSYSRST#	VMEbus Reset Input	Asserts LRST# on the local bus, resets the Universe, and reconfigures power-up options.
VME Output	VXSYSRST	VMEbus System Reset	Universe output for SYSRST* (resets the VMEbus)
PCI Input	PWRRST#	Power-up Reset	Resets the Universe and reconfigures power-up options.
	RST#	PCI Reset Input	Resets the Universe from the PCI bus.
	VME_RESET#	VME Reset Initiator	Causes Universe to assert VXSYSRST
PCI Output	LRST#	PCI Bus Reset Output	Resets PCI resources

While the Universe provides register support for resets (Table 2-21), the Universe is only reset by reset input pins. In order to reset the Universe by writing to registers, the Universe reset outputs must be connected to the Universe reset inputs. For example, the SW_LRST bit in the MISC_CTL register, which asserts the LRST# output, will not reset the Universe itself unless LRST# causes an assertion of RST#. Caution must be exercised in order not to connect the Universe in a loop such that the Universe may be driven in a permanent reset.

Table 2-21 Universe Register Support for Resets

Register and Table	Name	Type	Function
MISC_CTL Table A-60	SW_LRST	W	Software PCI Reset 0=No affect, 1=Initiate LRST# A read always returns 0.
	SW_SYSRST	W	Software VMEbus SYSRESET 0=No affect, 1=Initiate SYSRST* A read always returns 0.
VCSR_SET Table A-86	RESET	R / W	Board Reset Reads: 0=LRST# not asserted, 1=LRST# asserted Writes: 0=no effect, 1=assert LRST#
	SYSFAIL	R / W	VMEbus SYSFAIL Reads: 0=VXSYSFAIL not asserted, 1=VXSYSFAIL asserted Writes:0=no effect, 1=assert VXSYSFAIL
VCSR_CLR Table A-85	RESET	R / W	Board Reset Reads: 0=LRST# not asserted, 1=LRST# asserted Writes: 0=no effect, 1=negate LRST#
	SYSFAIL	R / W	VMEbus SYSFAIL Reads: 0=VXSYSFAIL not asserted, 1=VXSYSFAIL asserted Writes:0=no effect, 1=negate VXSYSFAIL

More detailed information about the effects of various reset events is provided in the next section.

2.12.1.2 Universe Reset Circuitry

Figure 2-20 below details the categories of effects of each reset event. For example, it shows that when the PWRRST# signal is asserted, the VME Services, Clock Services, General Services, UCSR and power-up options are all reset, while VXYSRST# and LRST# are asserted. Figure 2-20 lists the members of these categories of reset effects. For example, the VME Services category is composed of the VME Arbiter, VME Bus Timer, and VCSR registers. Thus, when PWRRST# or VRSYSRST# is asserted, all of the said VME Services are reset.

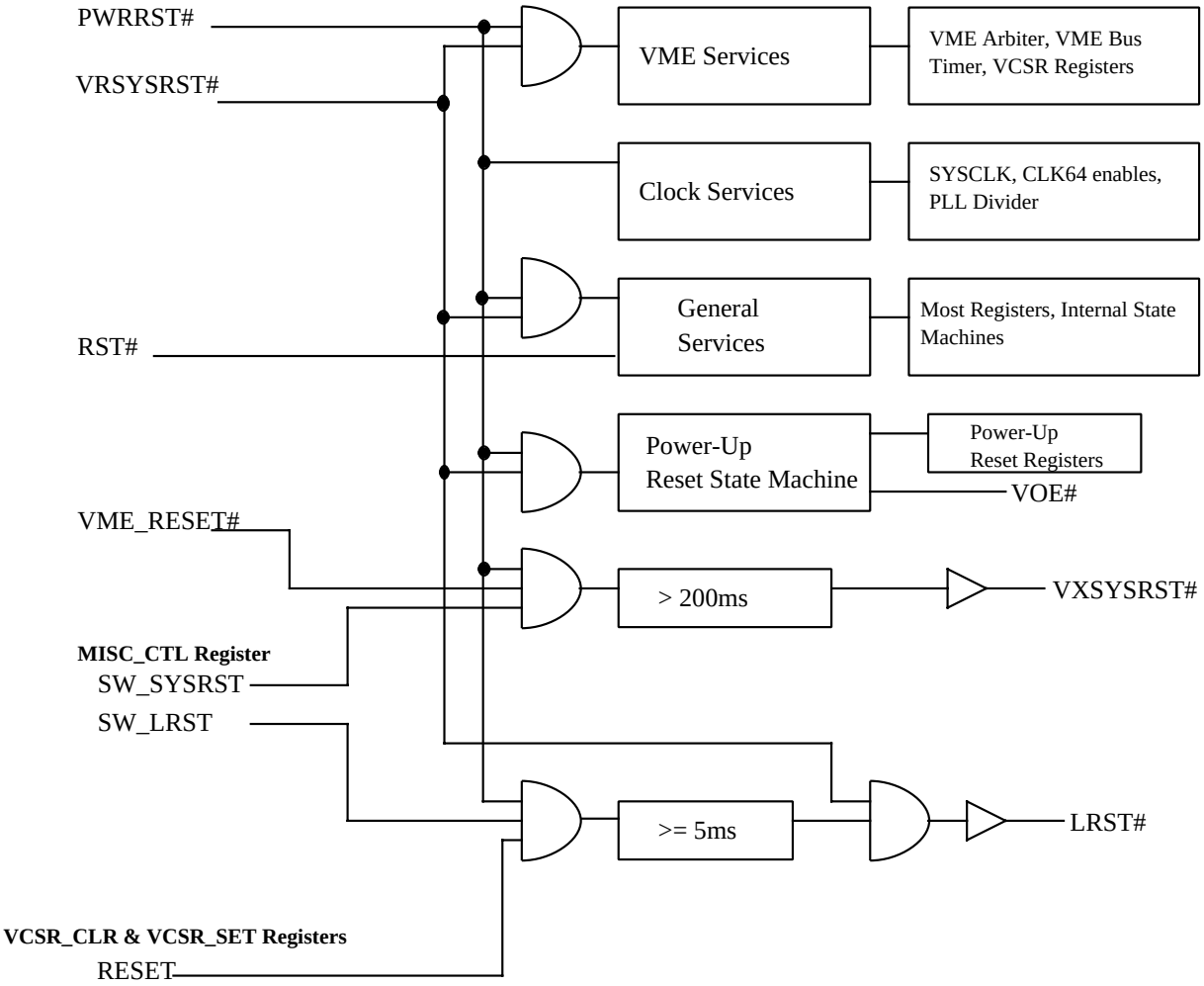


Figure 2-20 Functional Block Diagram of Reset Circuitry

Figure 2-20 above also indicates the reset effects that are extended in time. For example, VXSYSRST# remains asserted for 256 ms after all initiators are removed - this satisfies VMEbus rule 5.2 (minimum of 200 ms SYSRST*). The external 64 MHz

clock controls this assertion time. LRST# is asserted for 5ms or more from all sources except VRSYSRST#.

Figure 2-20 also illustrates the effects of register events. For example, it shows how asserting the SW_LRST bit in the MISC_CTL register causes LRST# to be asserted for at least 5ms.

The interface option includes configurable jumpers for enabling / disabling and driving / receiving VMEbus SYSRESET*. Refer to Section 2.2 on page 2-3.

2.12.2 Power-Up Options

The majority of the Universe power-up options (listed below) are loaded from the VMEbus address and data lines after any PWRRST# or SYSRST* event. Two power-up options are not initiated by PWRRST#. The first of these is PCI bus width (a power-up option required by the PCI bus specification), and this is loaded on any RST# event from the REQ64# pin. The second special power-up option is VMEbus SYSCON enabling, required by the VMEbus specification. The SYSCON option is loaded during a SYSRST* event from the BG3IN* signal.

All power-up options are loaded during their particular reset event from the state of a particular pin or group of pins. Each of these pins has internal pull-downs to put the Universe into a default configuration. If other than the default is required, a pull-up of approximately 10kΩ (or active drive) is required on the signal. These pull-ups are only necessary on the general group of power-up options since the PCI and VMEbus specifications provide for the appropriate levels to be actively driven on the REQ64# and BG3IN* signals, respectively.

The pull-ups for the general power-up options (if other than default values are required) must be placed on the VA[31:1], VLWORD, and VD[31:27] lines. During reset, the Universe will negate VOE#, putting these signals into a high-impedance state. The pull-ups (or internal pull-downs) will bring the option pins to their appropriate state. Within two CLK64 periods after both SYSRST* and PWRRST# are negated, the Universe latches the levels on the option pins, and then negates VOE# one clock later. This enables the VMEbus transceivers inwards.

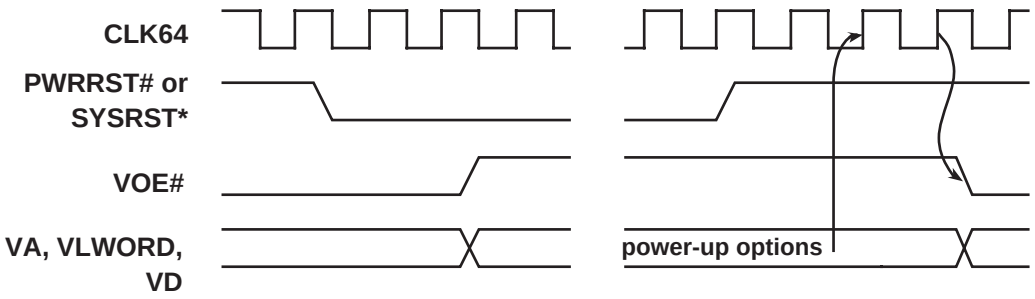


Figure 2-21 Power-up Options Timing

Note

Note that because of the operation of the power-up options, the VME buffers are not enabled until three CLK64 periods after release of SYSRST* (approximately 45ns). Including worst case backplane skew of 25ns implies that the Universe will not be prepared to receive a slave access until 70ns after release of SYSRST*.

2.12.2.1 Power-up Option Descriptions

PCI Bus Width

The Universe's PCI interface can be used as a 32-bit bus or 64-bit bus. If used as a 32-bit interface, the 64-bit pins, AD[32:63] and ACK64# are left unterminated (REQ64# must be pulled-up at reset for the Universe to be configured with a 32-bit PCI bus, see "32-Bit Versus 64-Bit PCI" on page 2-26). This power-up option, required by the PCI specification, provides the necessary information to the Universe so that these unused pins may be left unterminated.

Note

The VMIVME-7589 interface option supports only a 32-bit PCI bus.

SYSFAIL* Assertion

This power-up option causes the Universe to assert SYSFAIL* immediately upon entry into reset. The SYSFAIL* pin is released through a register access. Note that this power-up option is over-ridden if VME64 Auto-ID has been enabled. This option would be used when extensive on-board diagnostics need to be performed before release of SYSFAIL*. After completion of diagnostics, SYSFAIL* may be released through software or through initiation of the VME64 Auto-ID sequence if that mechanism is to be used.

Table 2-22 Power-Up Options

Option	Enabled by	Register	Field	Default	Pins
PCI Register Access	PWR, SYS	PCI_BS	SPACE	Jumper Configuration (1)	VA[1]
SYSFAIL* Assertion	PWR, SYS	VCSR_SET	SYSFAIL	Jumper Configuration (1)	VD[27]
		VCSR_CLR	SYSFAIL		
Auto-Syscon Detect	SYS	MISC_CTL	SYSCON	enabled	BGIN[3]*
(1) Refer to Section 2.2 on page 2-3					

CHAPTER
3

Auxiliary Functions

Contents

The material in Chapter 3 is exclusively VMIC.

3.1 Auxiliary Bus Timeout Timer

The VMIVME-7589 VMEbus interface option contains a programmable bus timeout timer which functions only when the interface option is the system controller. The VMIVME-7589 VMEbus interface option also has an auxiliary bus timeout timer which can be enabled to function when the board is not the system controller. This timer is enabled by setting the BTO bit in the system COMM register located at the SYS_BASE offset 0x00. The timer has four programmable timeout periods which are set by the BTOV(1:0) bits in the system COMM register. The four settings are shown in Table 3-1.

Table 3-1 Auxiliary Bus Timeout Timer Settings

BTOV1	BTOV0	Timeout Period (PCI CLK = 25,30,33 MHz)			
0	0	21	18	16	µs
0	1	85	70	64	µs
1	0	341	282	256	µs
1	1	1.33	1.09	1.00	ms

As shown above, the timeout period is a function of the PCI clock frequency. The auxiliary timer should NOT be enabled when the VMIVME-7589 VMEbus interface option is the system controller.

3.2 Auxiliary BERR Interrupt

The VMIC interface board contains logic which is capable of masking BERR*s from the Universe chip. The first release of the Universe chip contains two erratas which are capable of locking the VMEbus in response to a VMEbus BERR*. In both instances, there is no clean way to recover from the lock-up. The auxiliary BERR handler was added to avoid the erratas and, consequently, the VMEbus lock-up.

The circuit works as follows. The logic is enabled by setting the ABLE bit in the COMM system register loaded at SYS_BASE offset 0x00. Once enabled, the logic masks any BERR* from the Universe chip and supplies DTACK* instead. The logic prevents the Universe from entering the errata state and potentially locking the VMEbus.

The logic also captures the BERR* address and address modifier and sets a BERR status bit in the B_INT_STATUS system register located at SYS_BASE offset 0x04. The logic will hold the BERR status until the error is processed by writing a 1 to the bus error (BERR) bit of the B_INT_STATUS register. This should be done only after the application has read the B_INT_STATUS register for the address modifier code and the B_Address register for the address. The logic can also be enabled to generate a PCI INTA# or SERR#.

Also see Section 2.8.4 Auxiliary BERR Interrupt Generation on page 2-62.

CHAPTER

4

Endian Conversion

Contents

The material in Chapter 4 is exclusively VMIC.

4.1 VMEbus Byte Lanes

For a given addressing mode, the VMEbus specification defines various types of data transfer cycles to access 1-, 2-, 3-, or 4-byte locations at once. A set of four adjacent byte locations differing only in address bits (A00, A01) is defined as a four-byte group, or a “byte (0-3)” group. Address lines A02-A31 select a four-byte group, then four additional addressing lines (DS0*, DS1*, A01, and LWORD*) select which byte(s) within the group are accessed.

Table 4-1 depicts the Even / Odd Byte assignments to the VMEbus data lines.

It is important to note the major byte-ordering differences between the VMEbus and the Intel Pentium® microprocessor. In addition, communication between Motorola-compatible 680X0 VMEbus modules and the VMIVME-7589 VMEbus interface option requires special attention to avoid byte-ordering conflicts.

Table 4-1 VMEbus Byte Assignment to the Data Lines

DTB CYCLE TYPE	D31-D24	D23-D16	D15-D08 EVEN ADDRESS	D07-D00 ODD ADDRESS
D08(EO) EVEN OR ODD				
Single Odd Byte(3)				Byte(3)
Single Even Byte(2)			Byte(2)	
Single Odd Byte(1)				Byte(1)
Single Even Byte(0)			Byte(0)	
D08(O) ODD ONLY				
Single Odd Byte(3)				Byte(3)
Single Odd Byte(1)				Byte(1)
D16				
Double Byte(2-3)			Byte(2)	Byte(3)
Double Byte(0-1)			Byte(0)	Byte(1)
D32				
Quad Byte(0-3)	Byte(0)	Byte(1)	Byte(2)	Byte(3)

4.2 Byte Ordering: Big Endian / Little Endian

The byte-ordering issue exists due to the different traditions at the major microprocessor manufacturers, Motorola and Intel. Much VMEbus equipment is designed around Motorola's 680X0 processors and compatibles, which store multiple-byte values in memory with the *most* significant byte at the lowest byte address. This byte-ordering scheme became known as "big-endian" ordering. On the other hand, Intel's 80X86 microprocessors, upon which MS-DOS® is based, store multiple-byte values in memory with the *least* significant byte in the lowest byte address, earning the name "little-endian" ordering.

The VMIVME-7589 VMEbus interface option uses an Intel Pentium® microprocessor, which uses little-endian byte ordering. Byte arrangement and the byte relationship between data in the processor and transferred data in memory are shown in Figure 4-1.

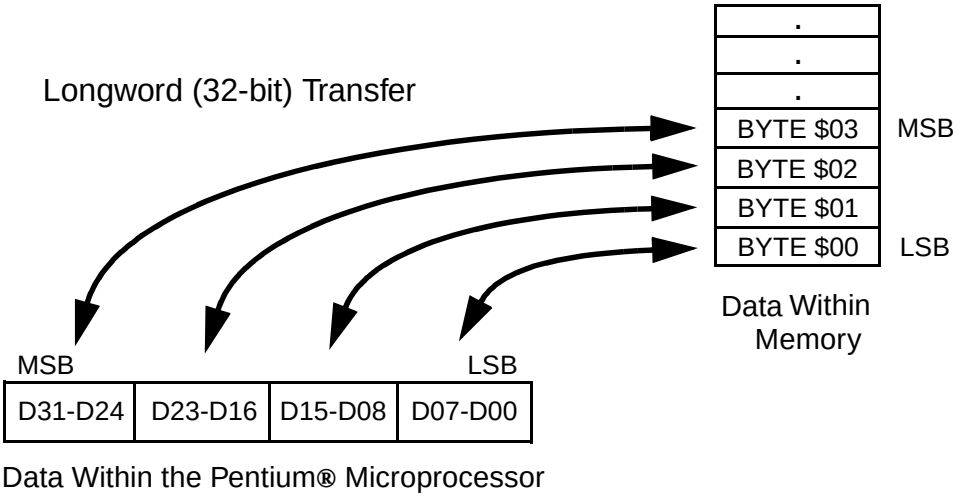


Figure 4-1 Byte Relationships Using the Little-Endian Pentium® Microprocessor

Note that in little-endian processors like the Pentium®, the processor's least significant byte is stored in the lowest byte address after a multiple-byte write (such as the longword transfer illustrated), while the processor's most significant byte is stored in the highest byte address after such transfers. Conversely, the processor considers data retrieved from the lowest byte address to be the least significant byte after a multiple-byte read. Data retrieved from the highest byte address is considered to be the most significant byte.

Contrast the behavior of the little-endian Pentium® in Figure 4-1 with the same longword transfer using a big-endian processor like the Motorola 68040 in Figure 4-2 below.

Note that the big-endian 68040 handles the same longword transfer in a completely different manner than the little-endian Pentium® microprocessor. During a multiple-byte transfer like the longword transfer illustrated, a big-endian processor writes its least significant byte in the highest byte address in memory, while its most significant byte is written to the lowest address. The converse is true during read operations: the data in the lowest byte address is considered to be the most significant, while the byte in the highest address is considered to be the least significant.

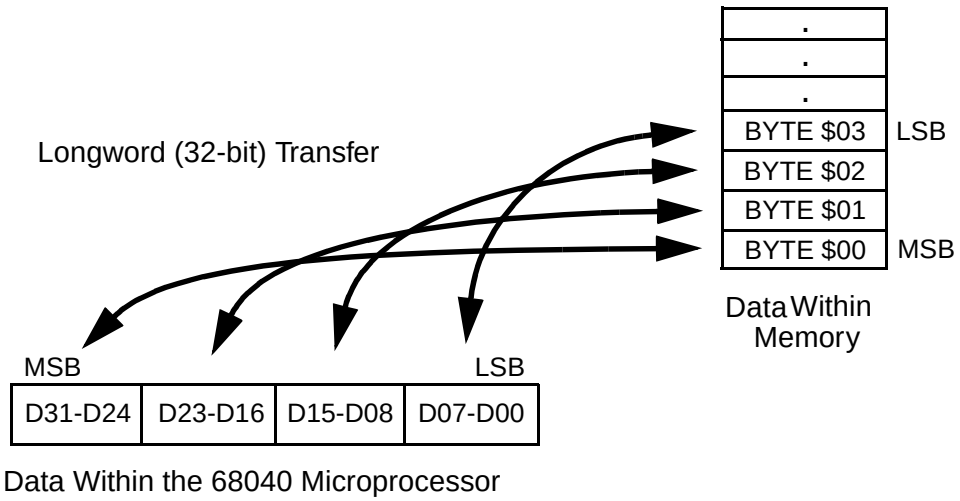


Figure 4-2 Byte Relationships Using the Big-Endian 68040 Microprocessor

The VMEbus Specification does not specify which byte of a multiple-byte transfer is most significant. The VMEbus Specification does, however, require certain byte lanes to be associated with certain byte addresses. As shown in Table 4-1 on page 4-2, byte(0) must be transferred on data lines D31-D24 during a longword transfer while byte(3) must be transferred on lines D7-D0. This byte and address alignment is exactly the same as that for a big-endian processor such as the Motorola 68040 in Figure 4-2.

If a little-endian Pentium® were to have its data bus directly connected to the VMEbus (i.e., D31 to D31, D30 to D30, etc.), then the most significant byte data supplied to the VMEbus D31-D24 byte lane during a longword write would be stored by the VMEbus in the lowest of the four destination byte addresses – opposite that expected by the Pentium® microprocessor (see Figure 4-1). This poses no problem if the 32-bit value written is always read back using a similar longword transfer (i.e., all four bytes at once), since the swapped data gets swapped again and appears to the Pentium® microprocessor exactly as it should. However, if the data written by the 32-bit longword transfer were to be retrieved using any other method, for example, using four separate byte transfers creates a problem. The data at the lowest byte address would be incorrectly assumed to be the least significant, while it is actually the most significant.

The problem cannot be solved by simply connecting the Pentium® microprocessor to the VMEbus with its byte lanes crossed. For example, the Pentium® microprocessor uses D0-D7 to transfer a byte to address $00, while the VMEbus requires D8-D15 be used. For this reason, special hardware has been incorporated into the VMIVME-7589 VMEbus interface option to facilitate different kinds of byte swapping for varying circumstances.

4.3 Endian Conversion Hardware

The Universe chip performs Address Invariant translation between the PCI and VMEbus interfaces. Address Invariant mapping or “Non-endian conversion” mode maintains the byte ordering between the two interfaces (i.e. data originating in little-endian mode on the CPU / PCI side will remain in little-endian mode on the VMEbus side of the interface). However, the VMIVME-7589 VMEbus interface option has external endian conversion logic which allows the application to perform independent master / slave hardware endian conversion. The enables for the circuitry (MEC,SEC) are in the system COMM register located at the SYS_BASE offset 0x00.

4.4 Unaligned Transfers with Endian Conversion Enabled

The VMIVME-7589 VMEbus interface option was designed to be consistent with our endian conversion approach that was used on the similar Cypress VIC064-based interface bridge. That approach is: defeat endian conversion for unaligned transfers to / from VMEbus - two cases of 3-byte, one unaligned 2-byte case.

The defeat on unaligned transfers (UAT) philosophy was adopted given that data sharing between CPUs with different endianess only makes sense for longword aligned and word aligned data. However, the Universe chip breaks up 3-byte transfers on the PCI bus into two VMEbus transfers: a byte access and a word access. Our endian conversion hardware which resides on the VMIVME-7589 VMEbus interface option of the Universe chip “sees” the word access (which is word aligned) and performs a byte swap. This causes a word within the 3-byte entity to be swapped. To avoid this inadvertent swapping, the user should disable master endian conversion when performing any unaligned transfers to the VMEbus. This only affects transfers originating from the Pentium® microprocessor and destined for the VMEbus.

4.5 PCI Bus Data Combining: Byte Swap

The VMIVME-7589 VMEbus interface option performs endian conversion (byte swapping) based on size. If a longword access is performed, it swaps the two words and the two bytes within those words. If a word access is performed, it swaps the two bytes. If byte access is performed, no swapping occurs.

Unfortunately, successive writes from the Pentium® microprocessor to VME through the host bridge are subject to combining: four bytes may combine to form one longword, two words may combine to form a longword, two bytes may combine to form a word. Since the endian conversion logic swaps on size, the data combining causes data to be swapped when it shouldn't have been, or longword swaps to occur when word swaps should have occurred.

This phenomena only occurs for Pentium® microprocessor to VME write accesses: reads are not affected, nor are VME slave accesses or DMA BLTs.

The data combining can be defeated by programming the Universe VME master channel (PCI slave channel) to be longword, word or byte wide depending on the size of accesses being generated from the Pentium® microprocessor. Data combining may still occur on the PCI bus, but the Universe chip will only generate the programmed size transfers on VMEbus. Thus, the endian conversion logic will "see" the data in the size that is consistent with swap protocol. Again, reads, VME slave accesses in either direction, or DMA BLTs are not affected.

CHAPTER

5

Errata

The material in Chapter 5 is exclusively VMIC.

The following is a list of current Universe errata that must be considered when programming the VMIVME-7589 VMEbus interface option:

1. The user should not enable the FAIR request mode in the MAST_CTL register. The default setting is disabled. Refer to Universe Device errata No. 2.
2. The Universe register space is defined as 4 K; however, the actual size is 64 K. The user should be careful not to map a valid slave image within the 64 K Universe address space; otherwise, the registers may be overwritten. Refer to Universe Device errata No. 7.
3. The user should disable the Universe PCI master retry counter by programming the MAXRTRY bits to 0000 in the MAST_CTL register. The default setting for the counter is enabled. Refer to Universe Device errata No. 10.
4. The user should program the Universe PCI aligned burst size in the MAST_CTL register to 32 bytes when performing DMA transfers from the PCI bus to the VMEbus (i.e. read PCI ... write VMEbus). The default for the PCI aligned burst size is 32 bytes. Refer to Universe Device errata No. 11.
5. The user should not enable posted writes in the Universe PCI slave channels. The default setting is posted write enable (PWEN) disabled. Refer to Universe Device errata No. 13.
6. If the user chooses to enable posted writes in the PCI slave channels, the external BERR logic should be enabled to avoid a potential lock of the VMEbus. The default setting for the external BERR logic is disabled. Refer to Universe Device errata No. 1, and No. 13.
7. The user should not generate VMEbus RMW cycles by way of the Universe special cycle generator logic. If VMEbus RMW cycles are required, use the VMEbus ownership bit to gain exclusive access to the VMEbus, generate necessary transactions on the VMEbus, and release the VMEbus ownership bit. Refer to Universe Device errata No. 15.
8. If the user enables the DMA channel to move data from the PCI bus to the VMEbus, the auxiliary BERR logic should be enabled to avoid a potential lock of the VMEbus. The default setting for the external BERR logic is disabled. Refer to Universe Device errata No. 14.

The references to errata are defined in the “Tundra CA91C042 Universe Devise Errata” document available from Tundra Corporation.

CHAPTER

6

PCI/VMEbus Deadlock

Contents

The material in Chapter 6 is exclusively VMIC.

6.1 Scenario Overview

There is a deadlock scenario which is inherent to systems containing a PCI/VME interface. The PCI specification allows for and defines devices called bridges. A bridge could be a host bridge which interfaces a host CPU to the system PCI bus. Another type of bridge is the PCI-to-PCI bridge which bridges two different PCI busses.

In order to optimize system performance, PCI specification allows bridges to provide buffering. Almost all host bridges and PCI-PCI bridges contain write-posting buffers. These buffers allow writes from one side of the bridge to be acknowledged before the data is actually written to the other side of the bridge.

Without write posting, the CPU or PCI initiator would have to wait for the device that is receiving the data to acknowledge the data transfer before it could proceed to the next bus transaction. In addition to allowing for write posting, the PCI bridge specification imposes transaction ordering rules on the bridge design to ensure data consistency in the system. These ordering rules are imposed on host bridges and on PCI-to-PCI bridges. One ordering rule is that reads cannot traverse across a bridge until all write-posted data has been flushed. In a PCI/VME system, this ordering rule coupled with the lack of retry on the VMEbus create the potential for system deadlocks.

6.2 An Example

To understand the need for ordering rules and the potential for deadlock, consider examine the following scenario.

Suppose the CPU writes a buffer to a VMEbus SRAM board, and then sets a flag in the VMIVME-7589 DRAM. Meanwhile, a VME master, such as a 68 K CPU, is polling the flag in DRAM, to determine when the SRAM buffer is ready to process. The ordering rule guarantees that the VMEbus master will not “see” the status flag set in DRAM until all the VMEbus write data has been flushed from the host bridge. However, a deadlock condition will occur if the PCI / VME interface has been granted the PCI bus to perform a read from DRAM while write posting data destined for the VMEbus is contained in the host bridge. In this case, the write posted data cannot flush (complete on VME) since the VMEbus is being held by the 68 K CPU, which cannot complete its read because the host bridge will not allow the read to occur, thus a DEADLOCK occurs.

The above deadlock scenario is inherent in PCI systems that interface to busses that do not have retry capability. Since the VMEbus does not currently provide for bus retry, the resulting deadlock condition will result in a VMEbus error (BERR) condition.

6.3 Possible Solutions

The simplest solution is to never simultaneously enable the VMEbus master and slave interfaces; however, in most systems this is not practical.

If simultaneous VMEbus master / slave interfacing is required (i.e. shared memory applications), the user can use the Universe VMEbus ownership bit in the MAST_CTL register to guarantee exclusive access to the VMEbus. The user would employ the following protocol when using the VME ownership bit:

1. Set the VME ownership bit in the MAST_CTL register.
2. Poll the VOWN_ACK bit which is asserted when the VMEbus has been acquired.
3. Once the VOWN_ACK bit is asserted, perform VMEbus master write operations.
4. Clear the VME ownership bit.

Note that the assertion of the VOWN_ACK bit can be programmed to generate a PCI interrupt.

APPENDIX

A

Registers

The PCI Target Configuration Header is illustrated in Figure A-1 on page A-2.

The System Registers provide control / status not contained within the Universe chip such as master / slave endian conversion and non-slot 1 bus timeout timer. Table A-2 on page A-4 defines the System Register Map.

The Mailbox Register Map for the Universe-based interface in included in Table A-3 on page A-6.

The Auxiliary Functions Global Interrupt Mask Map is included in Table A-4 on page A-8.

The Universe Control and Status Registers (UCSRs) facilitate host system configuration and allow the user to control Universe operational characteristics. The UCSRs are divided into four groups:

- the DMA Channel,
- PCI Configuration Space (PCICS),
- VMEbus Control and Status Registers (VCSR), and
- Universe Device Specific Status Registers (UDSR).

The Universe registers are little-endian. The register map for the UCSR's is included in Table A-4 on page A-8.

Note

Appendix A is a combined Tundra / VMIC chapter. VMIC information is included in:

- the auxiliary function global interrupt mask register,
- the system register section, and
- the mailbox register section.

The Universe Control and Status Register material is a direct reprint of Tundra-provided information.

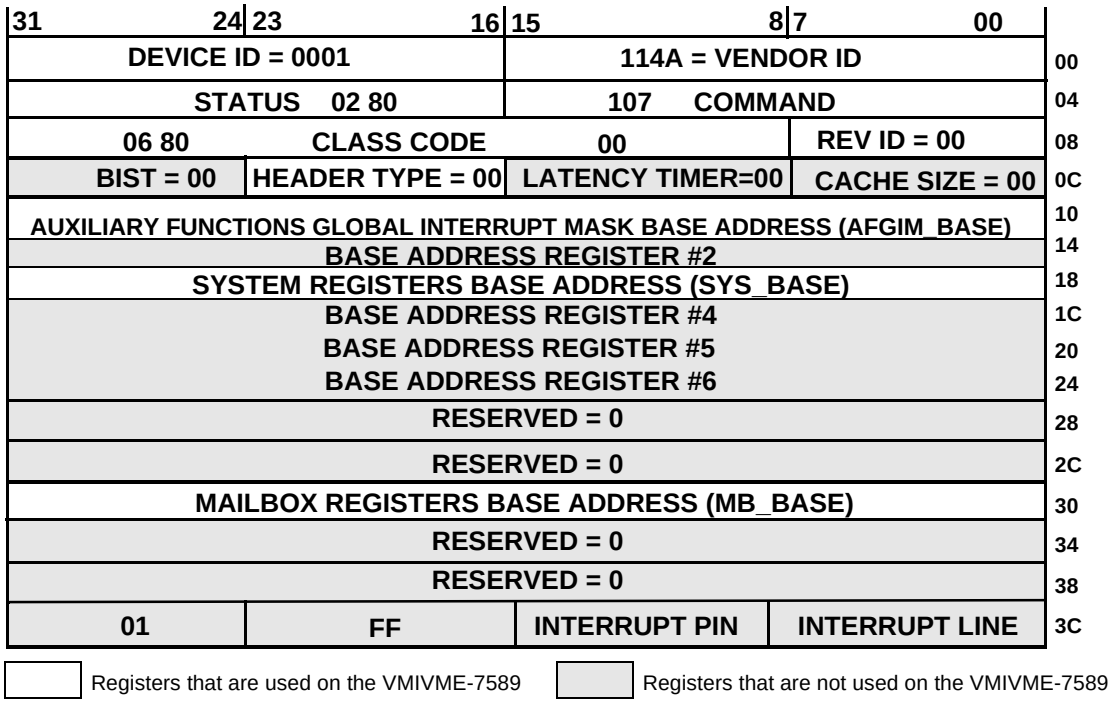


Figure A-1 (PLX9060ES) PCI Target Configuration Header

Table A-1 Auxiliary Functions Global Interrupt Mask - (Offset: 0x68)

Field	Description	Read	Write	Value After Reset
0	Not Used - Not Supported	NA	NA	0
1	Not Used - Not Supported	NA	NA	0
2	Not Used - Not Supported	NA	NA	0
7:3	Not Used - Not Supported	NA	NA	0
8	PCI interrupt enable. A value of 1 will enable PCI interrupts.	Yes	Yes	1
9	Not Used - Not Supported	NA	NA	0
10	Not Used - Not Supported	NA	NA	0
11	PCI local interrupt enable. A value of one will enable a local interrupt input to generate a PCI interrupt. Use in conjunction with PCI interrupt enable. Clearing the local bus cause of the interrupt will clear the interrupt.	Yes	Yes	0
12	Not Used - Not Supported	NA	NA	0
13	Not Used - Not Supported	NA	NA	0
14	Not Used - Not Supported	NA	NA	0
15	Not Used - Not Supported	NA	NA	0
16	Not Used - Not Supported	NA	NA	0
17	Not Used - Not Supported	NA	NA	0
19:18	Not Used - Not Supported	NA	NA	0
20	Not Used - Not Supported	NA	NA	0
22:21	Not Used - Not Supported	NA	NA	0
23	Not Used - Not Supported	NA	NA	0
24	Not Used - Not Supported	NA	NA	0
26:25	Not Used - Not Supported	NA	NA	0
27	Not Used - Not Supported	NA	NA	0
31:28	Not Used - Not Supported	NA	NA	0

Table A-2 System Register Map

Register Name	Mnemonic	Access	Offset																
Command	COMM	W: R/W	0x00	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
				X	X	X	X	VME_EN	MB_M3	MB_M2	MB_M1	MB_M0	BERRIM	BTOV1	BTOV0	BTO	ABLE	SEC	MEC
BERR Interrupt Status	B_INT_STATUS	B,W,L: R/WC	0x04	**D31**	**D30**	**D29**	**D28**	**D27**	**D26**	**D25**	**D24**	**D23**	**D22**	**D21**	**D20**	**D19**	**D18**	**D17**	**D16**
				X	X	X	X	X	X	X	X	X	X	AM[5]	AM[4]	AM[3]	AM[2]	AM[1]	AM[0]
				D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	BERR_S
BERR Interrupt Mask	B_INT_MASK	B: R/W	0x08	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
				X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	BERR_M
BERR Address Log	B_ADDRESS	B,W,L: R	0x0C	**D31**	**D30**	**D29**	**D28**	**D27**	**D26**	**D25**	**D24**	**D23**	**D22**	**D21**	**D20**	**D19**	**D18**	**D17**	**D16**
				A31	A30	A29	A28	A27	A26	A25	A24	A23	A22	A21	A20	A19	A18	A17	A16
				D15	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
				A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	0

Offset: Offest from SYS_BASE (PLX config register 0x18)

The Table A-2 System Register Map Bit Definitions are defined as follows:

COMM (8-bit Read/Write register):

- MEC: Master Endian Conversion enable bit (1=enabled, **0=disabled**)
- SEC: Slave Endian Conversion enable bit (1=enabled, **0=disabled**)
- ABLE: Auxiliary BERR logic enable bit (1=enabled, **0=disabled**
- BTO: Auxillary Bus Timeout Timer enable bit (1=enabled, **0=disabled**)
- BTOV(1:0): Auxiliary Bus Timeout Timer value: (**00 = 16 µs,** 01 = 64 µs, 10 = 256 µs, 11 = 1 ms)
- BERRIM: BERR Interrupt map bit (1=PCI SERR#, **0=PCI INTA#**)
- MB_Mx: Mailbox X Interrupt mask bit (1=enabled, **0=masked**)
- VME_EN: VMEbus enable (1=enabled, **0=disabled**)

B_INT_STATUS (32-bit Read/WriteClear register):

- AM[x]: AM code of BERR cycle (Valid if BERR_S=1)
- BERR_S: BERR status bit (1=BERR detected, **0=No BERR detected**)

B_INT_MASK (1-bit Read / Write register):

- BERR_M: BERR interrupt mask (1=BERR interrupt enabled, **0=BERR interrupt disabled**)

B_ADDRESS (32-bit Read-only register):

- A(31:1): Address of BERR cycle (Valid if BERR_S=1)

Figure A-2 on page A-7 summarizes the supported register access mechanisms.

Table A-3 Mailbox Register Map

			PCI/VME																
Register Name	**Mnemonic**	**Access**	**Offset**	**D15**	**D14**	**D13**	**D12**	**D11**	**D10**	**D9**	**D8**	**D7**	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**
Mailbox 0	MB0	Byte R/W	0x01	X	X	X	X	X	X	X	MB0_INT	X	X	X	X	X	X	X	X
Mailbox 1	MB1	Byte R/W	0x05	X	X	X	X	X	X	X	MB1_INT	X	X	X	X	X	X	X	X
Mailbox 2	MB2	Byte R/W	0x09	X	X	X	X	X	X	X	MB2_INT	X	X	X	X	X	X	X	X
Mailbox 3	MB3	Byte R/W	0x0D	X	X	X	X	X	X	X	MB3_INT	X	X	X	X	X	X	X	X

Bit Definitions:
MBx**:**
MBx_INT: Mailbox x Interrupt bit (1=Generate PCI interrupt, **0=Clear PCI interrupt**)
Operation: VMEbus master sets this bit to generate a PCI INTA interrupt.
Host processor clears this bit as part of the interrupt handler to acknowledge interrupt.
X : Not Applicable, Reads back as a 1.

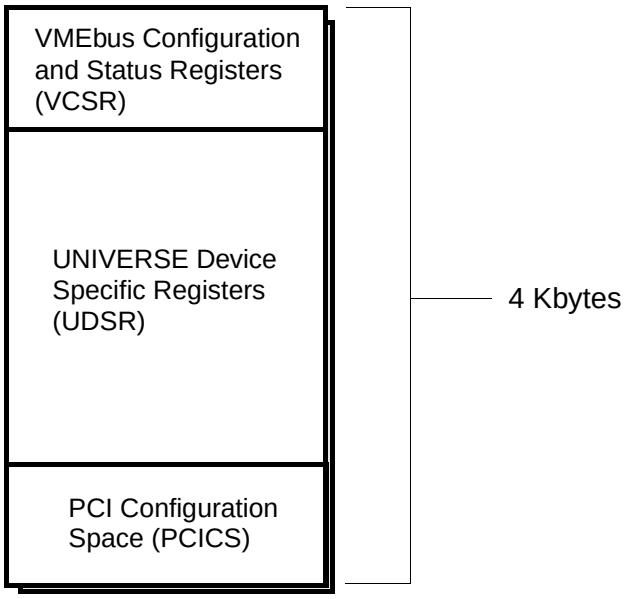


Figure A-2 UCSR Access Mechanisms

Table A-4 lists the Universe registers by address offset. The tables following the register map (Table A-5 to Table A-87) provide detailed descriptions of each register.

Address offsets in Table A-4 apply to accesses from the PCI bus and to accesses from the VMEbus side using the VMEbus Register Access Image (see “Registers” on page 2-92). 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" field 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" field 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" field in the register description

Table A-4 Universe Register Map

Offset	Register	Name
000	PCI Configuration Space ID Register	PCI_ID
004	PCI Configuration Space Control and Status Register	PCI_CSR
008	PCI Configuration Class Register	PCI_CLASS
00C	PCI Configuration Miscellaneous 0 Register	PCI_MISC0
010	PCI Configuration 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 Configuration 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	
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

Table A-4 Universe Register Map (Continued)

Offset	Register	Name
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
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

Table A-4 Universe Register Map (Continued)

Offset	Register	Name
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
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

Table A-5 PCI Configuration Space ID Register (PCI_ID)

Register Name: PCI_ID	**Offset:000**

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 identifier
VID[15:0]	R	all	10E3	Vendor ID - PCI SIG allocated vendor identifier

Table A-6 PCI Configuration 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 (Continued)

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-7 PCI Configuration Class Register (PCI_CLASS)

Register Name: PCI_CLASS	**Offset:008**

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 defined 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-8 PCI Configuration 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	Configuration Space Layout
LTIMER [7:3]	R/W	all	0	Latency Timer: The latency timer has a resolution of 8 clocks.

Table A-9 PCI Configuration Base Address Register (PCI_BS)

Register Name: PCI_BS	**Offset:010**

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 specifies 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 specified 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 configuration transaction or a VMEbus register access.

Table A-10 PCI Configuration 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-11 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 Configuration 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-12 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 specifies the lowest address in the address range that will be decoded.

Table A-13 PCI Slave Image 0 Bound Address Register (LSI0_BD)

Register Name: LSI0_BD	**Offset:108**

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-14 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 field 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-15 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 Configuration 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-16 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-17 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-18 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-19 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 Configuration 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-20 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-21 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-22 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-23 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 Configuration 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-24 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-25 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-26 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-27 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 specified address. Address-Only cycles are initiated by either read or write cycles.

Table A-28 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-29 Special Cycle Swap/Compare Enable Register (SCYC_EN)

Register Name: SCYC_EN	**Offset: 178**

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 defines the bits that will be involved in the compare and swap operations for VME RMW cycles.

Table A-30 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-31 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-32 PCI Miscellaneous Register (LMISC)

Register Name: LMISC	**Offset: 184**

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 (Infinite), 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 first 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 configuration time, or after disabling all PCI Slave Images.

Table A-33 Special PCI Slave Image (SLSI)

Register Name: SLSI	**Offset:188**

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 specifies 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 specifies 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 specifies 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 Specifies 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 Configuration Space, 11=Reserved

This register fully specifies an A32 capable special PCI slave image. The base is programmable to a 64 Mbyte alignment, and the size is fixed at 64 Mbytes. Incoming address lines [31:26] (in Memory, I/O, or Configuration space) must match this field 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 fields to specify the AM code and the maximum port size for each region. The PGM field is ignored for the portion of each region mapped to A16 space.

No block transfer AM codes are generated.

Table A-34 PCI Command Error Log Register (L_CMDERR)

Register Name: L_CMDERR	**Offset: 18C**

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-35 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 qualified by bit L_STAT of the L_CMDERR register.

Table A-36 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-37 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 specifies 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 first linked-list entry.

Table A-38 DMA PCI Bus Address Register (DLA)

Register Name: DLA	**Offset:208**

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-39 DMA VMEbus Address Register (DVA)

Register Name: DVA	**Offset:210**

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-40 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-41 DMA General Control/Status Register (DGCS)

Register Name: DGCS	**Offset: 220**

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

DGCS Description (Continued)

Name	Type	Reset By	Reset State	Function
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
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-42 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 field before proceeding to modify the linked list. After the Linked List has been modified the PCI Resource must clear the UPDATE field by writing a logic 0.

Table A-43 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-44 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	VIRQ*x* Interrupt Status/Clear 0: no VIRQ*x* Interrupt StatusID vector available 1: VIRQ*x* 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 reflect the status of the respective VMEbus pins (after filtering 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-45 PCI Interrupt Map 0 Register (LINT_MAP0)

Register Name: LINT_MAP0	**Offset: 308**

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.

WARNING

Do not map any VMEbus interrupt to LINT#(2-7). Programming registers LINT_MAP0 or LINT_MAP1 with mappings to LINT#(2) - LINT#(7) will cause damage to the VMIVME-7588.

Table A-46 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.

WARNING

Do not map any VMEbus interrupt to LINT#(2-7). Programming registers LINT_MAP0 or LINT_MAP1 with mappings to LINT#(2) - LINT#(7) will cause damage to the VMIVME-7588.

Table A-47 VMEbus Interrupt Enable Register (VINT_EN)

Register Name: VINT_EN	**Offset:310**

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=LINT*x* Interrupt masked 1 =LINT*x* Interrupt enabled

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

Table A-48 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-49 VME Interrupt Map 0 Register (VINT_MAP0)

Register Name: VINT_MAP0	**Offset: 318**

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-50 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-51 Interrupt STATUS/ID Out (STATID)

Register Name: STATID	**Offset: 320**

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 specified by STATID field 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-52 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 field 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 field may not contain a valid vector.

Table A-53 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-54 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-55 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-56 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-57 VIRQ6 STATUS/ID Register (V6_STATID)

Register Name: V6_STATID	**Offset: 338**

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-58 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-59 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 2-17. and “FIFO Operation and Bus Ownership” on page 2-86.
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 Configuration space. If the bus number of the PCI address (bits [23:16]) is equal to the BUS_NO field, then the Universe generates a Type 0 configuration cycle, otherwise Type 1 is generated.

Table A-60 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-61 Miscellaneous Status Register (MISC_STAT)

Register Name: MISC_STAT	**Offset: 408**

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 reflects 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-62 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-defined AM codes.

Table A-63 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	**WARNING:** This function must always be disabled. Always set to 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 Configuration 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-6 on page A-12).

Table A-64 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 specifies the lowest address in the address range that will be decoded.

Table A-65 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-66 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-67 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	**WARNING:** This bit must be set to 0 before the EN bit is set to 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 Configuration 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-6 on page A-12).

Table A-68 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 specifies the lowest address in the address range that will be decoded.

Table A-69 VMEbus Slave Image 1 Bound Address Register (VSI1_BD)

Register Name: VSI1_BD	**Offset: F1C**

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-70 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-71 VMEbus Slave Image 2 Control (VSI2_CTL)

Register Name: VSI2_CTL	**Offset: F28**

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	**WARNING:** This bit must be set to 0 or damage to the board may occur. 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 Configuration 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-6 on page A-12).

Table A-72 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-73 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-74 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-75 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	**WARNING:** This bit must be set to 0 or damage to the board may occur. 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 Configuration 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-6 on page A-12).

Table A-76 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-77 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-78 VMEbus Slave Image 3 Translation Offset (VSI3_TO)

Register Name: VSI3_TO	**Offset: F48**

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

VSI3_TO Description

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

Table A-79 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-80 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 specifies the lowest address in the 4 Kbyte VMEbus Register Access Image.

The base address specifies 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 field 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-81 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 Configuration 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 field determines the PCI bus command that will be generated when the cycle is mapped to the PCI bus.

Table A-82 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-83 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 modifier code and the state of the IACK* signal. The register contents are qualified by the V_STAT bit.

Table A-84 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 qualified by the V_STAT bit of the V_AMERR register.

Table A-85 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 defined in the VME64 specification. Note that the RESET bit can be written to only from the VMEbus.

Table A-86 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 defined in the VME64 specification. 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-87 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 specifies one of thirty-one available CR/CSR windows as defined in the VME64 specification. Each window consumes 512 Kbytes of CR/CSR space.

Index

E

F

I

J

L

M

N

P

R

S

T

U

V

W