

Spec Number: 950-1373-01  
Rev. 50  
Date: July 19, 1991



# **Sun-4M**

## **System Architecture**

Sun Microsystems, Inc.

2550 Garcia Avenue

Mountain View, Ca 94043

(415) 960-1300

## Products Rights Notice:

Copyright © 1991-2008 Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054, U.S.A. All Rights Reserved

You understand that these materials were not prepared for public release and you assume all risks in using these materials. These risks include, but are not limited to errors, inaccuracies, incompleteness and the possibility that these materials infringe or misappropriate the intellectual property right of others. You agree to assume all such risks.

THESE MATERIALS ARE PROVIDED BY THE COPYRIGHT HOLDERS AND OTHER CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS (INCLUDING ANY OF OWNER'S PARTNERS, VENDORS AND LICENSORS) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THESE MATERIALS, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Sun, Sun Microsystems, the Sun logo, Solaris, OpenSPARC T1, OpenSPARC T2 and UltraSPARC are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the U.S. and other countries. Products bearing SPARC trademarks are based upon architecture developed by Sun Microsystems, Inc. UNIX is a registered trademark in the U.S. and other countries, exclusively licensed through X/Open Company, Ltd. The Adobe logo is a registered trademark of Adobe Systems, Incorporated. Part of the products covered by these materials may be derived from the Berkeley BSD systems licensed by the University of California. Sun Microsystems, Inc. has intellectual property rights relating to technology embodied in the product described in these materials. This distribution may include materials developed by third parties who have intellectual property rights therein. Products covered by and information contained in these materials may be controlled by U.S. Export Control laws and may be subject to the export or import laws in other countries. Nuclear, missile, chemical biological weapons or nuclear maritime end uses or end users, whether direct or indirect, are strictly prohibited. Export or reexport to countries subject to U.S. embargo or to entities identified on U.S. export exclusion lists, including, but not limited to, the denied persons and specially designated nationals lists may be prohibited.

**TABLE OF CONTENTS**

1. System Features	7
2. Introduction	
2.1 Goals	7
2.2 Scope	7
2.3 Acknowledgements	7
2.4 How to Use this Document	8
2.5 Glossary	9
3. Address space	
3.1 Virtual Address Space: Processor Cores	11
3.2 Physical Address Space	13
4. Processor Core Register Definitions	
4.1 Module Control Register	17
4.2 Context Table Pointer Register	17
4.3 Context Register	17
4.4 Synchronous Fault Registers	18
4.5 Aynchronous Fault Registers	20
4.6 Reset Register	20
4.7 MBus Port Address Register	21
5. System Register Definitions	
5.1 System Control Registers	22
5.2 M-to-S Asynchronous Error Registers	23
5.3 Counter-Timer Registers and User Timers	24
5.4 Diagnostic Support	26
5.5 Memory Registers	27
5.6 VMEbus Interface Registers	30
5.7 Interrupt Related Registers	31
5.8 IOMMU Registers	34
5.9 SBus Slot Configuration Registers	35
6. Interrupts	
6.1 Interrupt Definitions	36
6.2 Sun-4M Interrupt Structure	37
6.3 Interrupt Level Assignment	38
6.4 Programming Notes on Error Reporting	38
7. Memory Management and Cache Coherence	
7.1 SPARC Reference MMU (SRMMU)	40
7.2 I/O Memory Management Unit (IOMMU)	44
7.3 Cache Coherence	47
7.4 Write Buffers	49
7.5 Memory Model	50
8. The I/O Cache (IOC)	
8.1 Overview of the IOC	51
8.2 Management of IOC	51
8.3 I/O Cache Tag Format	52
8.4 I/O Cache Data Format, Diagnostic Access	52
8.5 I/O Cache Data Block Diagram	52

9. I/O Devices	
9.1 Keyboard/Mouse Interface and Serial Ports	53
9.2 EPROM	53
9.3 TOD/NVRAM	53
9.4 Audio/ISDN	54
9.5 VMEbus Master Port	55
9.6 SBus Expansion Slots	55
9.7 Floppy Disk Controller	56
9.8 Auxiliary I/O register 0	56
9.9 Generic I/O and Auxiliary I/O register 1	56
10. DVMA and DMA Devices	
10.1 SCSI and Ethernet Interfaces	57
10.2 VME Slave Port	57
10.3 SBus Expansion Slots	57
10.4 Valid DVMA Physical Addresses	57
11. Main Memory	
11.1 Memory Overview	58
11.2 Programming Notes on Software Scrub	58
12. Resets	59
13. References	60
Appendix A.I: Board Features: Galaxy	61
A.I: System Specific Information: Galaxy	62
A.I.0: Galaxy Block Diagram	62
A.I.1 SBus Details	63
A.I.2 MBus Details	63
A.I.3 Memory Details	64
A.I.4 I/O Details and Options Supported	65
A.I.5 Implementation ID numbers	65
A.I.6 SCSI/Ethernet Interface Details	65
A.I.7 Bugs/Features	67
A.I.8 Board Partition	67
A.I.9 Official Product Designations	67
A.II: System Specific Information: Campus-2	68
A.II.0: Campus-2 Block Diagram	69
A.II.1 SBus Details	70
A.II.2 MBus Details	70
A.II.3 Memory Details	71
A.II.4 Implementation-Specific Registers and Bits	72
A.II.5 Implementation ID numbers	75
A.II.6 I/O and DVMA Details	75
A.II.7 Bugs/Features	81
A.II.8 Board Partition	82
A.II.9 Official Product Designations	82

B.I: Module Notes: Viking/NE	83
B.I.1    Module Overview	84
B.I.2    Cache Details	84
B.I.3    Cache Coherence, Store Buffers, and Memory Models	87
B.I.4    Module Registers: differences from core	90
B.I.5    MMU Details	92
B.I.6    ASI's Implemented	94
B.I.7    Module Control Space Address Map	94
B.I.8    IU PSR Number	94
B.I.9    Module-Specific Quirks	94
B.II: Module Notes: Ross 605/64K	96
B.II.1   Module Overview	97
B.II.2   Cache Details	97
B.II.3   Module Registers: differences from core	99
B.II.4   Additional Registers Specific to this Module	99
B.II.5   MMU Details	101
B.II.6   ASI's Implemented	102
B.II.7   Module Control Space Address Map	103
B.II.8   IU PSR Number	103
B.II.9   Module-Specific Quirks	103
B.II.10  Module Write Buffers	103
B.II.11  Exiting Boot State	103
B.III: Module Notes: Viking/E\$	105
B.III.1  Module Overview	106
B.III.2  Cache Details	106
B.III.3  Cache Coherence, Store Buffers, and Memory Models	111
B.III.4  Module Registers: differences from core	115
B.III.5  MXCC Registers (ASI = 0x2)	116
B.III.6  MMU Details	121
B.III.7  ASI's Implemented	123
B.III.8  Module Control Space Address Map	124
B.III.9  IU PSR Number	124
B.III.10 Module-Specific Quirks	124
B.IV: Module Notes: Ross 604/64K	125
B.IV.1   Module Overview	126
B.IV.2   Cache Details	126
B.IV.3   Cache Flushing	127
B.IV.4   Module Registers: differences from core	127
B.IV.5   Additional Registers Specific to this Module	128
B.IV.6   MMU Details	130
B.IV.7   ASI's Implemented	130
B.IV.8   Module Control Space Address Map	131
B.IV.9   IU PSR Number	131

B.IV.10	Module-Specific Quirks	131
B.IV.11	Module Write Buffers	131
B.IV.12	Exiting Boot State	132
B.V: Module Notes: Ross 605/128K		133
B.V.1	Module Overview	134
B.V.2	Cache Details	134
B.V.3	Cache Coherence	136
B.V.4	Module Registers: differences from core	136
B.V.5	Additional Registers Specific to this Module	138
B.V.6	MMU Details	139
B.V.7	ASI's Implemented	140
B.V.8	Module Control Space Address Map	141
B.V.9	IU PSR Number	141
B.V.10	Module-Specific Quirks	141
B.V.11	Module Write Buffers	141
B.V.12.	Exiting Boot State	142
B.V.13	Programming Notes on Multichip Operation	142

## 1. System Features

The Sun-4M Architecture is a system architecture for Sun workstations and servers incorporating SPARC processors and the SPARC Reference MMU. Some of the implementations are designed around the SPARC MBus. Different implementations of this architecture will support between one and four processors. The primary I/O bus is a Sun SBus. Primary page size is 4KB.

This architecture is an outgrowth of the Sun-4 Architecture, with changes reflecting the nature of SRMMU and multiple processors. Some functions that are related to the processor core (IU, FP, cache, MMU) are replicated per processor. Interrupt steering logic is new, with MP support.

*Section 2.3 provides a functional index to this document.*

The first implementations based on this architecture are Galaxy and Campus-2. Appendices A.I and A.II (respectively) give implementation details for these systems. Appendices B.I, B.II, B.III, B.IV, and B.V define the specifics of each type of MBus processor module supported.

## 2. Introduction

### 2.1 Goals

The purpose of an architecture is to provide a consistent model of a system as seen by the operating system and by device drivers. In an ideal world this model is exact, and a single operating system will run identically across all implementations of that architecture. Sun has traditionally had a more lenient approach to system architecture, with minor variations allowed between implementations. It is the goal of this document to provide a definition of the model that is consistent among implementations, with variations specified in cache management models, device options, and devices managed at the device driver level. Implementations that choose to introduce changes at a more fundamental level should do so with a full understanding of the software development and test implications.

### 2.2 Scope

The Sun-4M Architecture provides a system definition for Galaxy and Campus-2 and follow-on projects. It comprises a programmer's model for use in developing system software, and architectural description for understanding of the system for hardware developers.

Up to this point this has been a living document. With the release of revision 2.0 it is firm for those products described, and flexible for future 4M projects. As this document was being developed there were several corporate committees involved in defining system architecture for the next generation of Sun products. Among these products are Campus-2, Sundragon (not a 4M machine), and Galaxy, using processor cores based on Viking and Ross modules. The Comet Committee and the SPARC Multi-Processor Architecture Committee have served as forums for arriving at a common architecture for the various products. This architecture document is in compliance with the resolutions of those two committees.

### 2.3 Acknowledgements

Many people have contributed to this document. The author would especially like to thank Tim Bucher, Steve Kleiman, Michel Cekleov, Jean-Marc Frailong, Ken Okin, and Frank Spies for their insights and often entertaining ideas. Also, thanks to the members of the Comet Committee, the MP SPARC Architecture Committee, the SBus Spec. Committee, the SPARC MBus Committee, the MBus Module Committee, the Galaxy design team, the ad-hoc MP Graphics Committee, the Comet write-buffer subcommittee, the ad-hoc I/O Model Committee, the Maxiray design team, the Viking design team, and of course the Sun Recycling Committee for processing the previous versions of this worthy document.

## 2.3 How to Use this Document

This document describes a system architecture for several system implementations and several processor core implementations (modules). This section is intended to help the reader in locating information which may be scattered across several sections.

### 2.3.1 Modules

Section 3 contains the generic module programmer's model, including virtual addresses, ASI's, and module register addresses. Section 4 contains the descriptions of generic module registers and their behavior. The 'B' appendices contain module-specific information such as cache size and style, extra registers implemented on each module, and extra control/status bits marked as 'reserved' in section 4. Section 7 discusses memory management and cache coherence, including the SPARC Reference MMU. Section 12 defines reset activity.

### 2.3.2 System Registers

System register addressing is described in section 3.2. Each register is described in detail in section 5. Registers associated with I/O are described in section 9 (for slave-only devices) and section 10 (for DVMA devices).

### 2.3.3 Memory Management and Cache Coherence

Chapter 7 is the bible for memory management and cache management. 7.1 defines the SPARC Reference MMU (SRMMU), and section 4 defines the behaviour of the SRMMU registers in Sun-4M implementations. 7.2 defines the IOMMU and explains how to manage it. Section 7.3 explains the MP and DVMA coherence support and rules. 7.4 details where write buffers may be implemented in the system and how to manage them. The I/O Cache is described in chapter 8, and the management of the IOC is also explained in that chapter.

### 2.3.4 SBus

SBus slot addressing is defined in 3.2.3, and SBus/IOMMU control space addressing in 3.2.2.4. The SBus Slot Configuration Registers are described in 5.9. SBus slots can each behave as both a master and a slave; the slave attribute is described in 9.6, and the master in 10.3.

### 2.3.5 VMEbus

Top-level addressing for the VME master port is described in 3.2.1, and VME control space in 3.2.2.3. VME interface registers are described section 5.6. Table 7.2.2 shows how VME addresses map to DVMA addresses. The I/O Cache is defined in section 8; it is used only for VME DVMA. The VME Master Port is defined in 9.5, and the VME slave port is defined in 10.3.

### 2.3.6 Interrupts

Section 6 defines how interrupts are distributed in the multi-processor Sun-4M architecture. 6.3 gives the interrupt level assignment for interrupts from hardware devices. Section 5.7 defines the registers associated with the interrupt logic and describes their behaviour.

### 2.3.7 Main Memory

Main memory is described in section 11. The memory registers and ECC are described in section 5.5.



## 2.4 Glossary

This section defines some technical terms that the reader may not be familiar with.

**Block:** The information that goes into a cache line. Many different blocks may map to the same line. In Sun-4M the block size is 32 bytes.

**Cache:** a small, fast memory that keeps recently accessed data, instructions, or translations local to the data sink

**Coherence, consistency:** keeping multiple cached copies of information up-to-date even when one copy gets modified (written to)

**Demap:** Changing the correspondence between a virtual and physical address, and purging stale cached translations

**DVMA:** Direct Virtual Memory Access; access from non-processor masters to memory or I/O devices via a memory management unit.

**Expansion:** Generally refers to options that a user can add to a system by plugging a circuit board into a backplane or into a socket on the CPU motherboard.

**Flush:** Purging of stale references from a cache in order to maintain consistency, used when the system software is changing the use of a virtual address or of an I/O Cache line.

**Harvard architecture:** An architecture where instructions and data use separate caches.

**Line:** An entry in a cache, which contains one or more blocks

**Module:** A *logical* module consists of a SPARC IU, FPU, cache(s), an SRMMU, and an interface to the system. A *physical* module (MBus module) is a daughterboard that plugs into a CPU that contains one or two *logical* modules. In general references to a module in this document are for *logical* modules unless otherwise specified.

**MMU:** Memory Management Unit; a device that translates processor or DVMA *virtual* addresses to *physical* addresses; may include access permission checks and status of page access. Two types of MMU are described in this specification, the SPARC Reference MMU and the IOMMU.

**MP:** Multiple Processor

**Mutex:** MUTual EXclusion lock; this is a semaphore in memory used by software to guarantee exclusive access to memory variables or to a hardware device, among multiple processors or threads

**On-board:** Generally refers to devices that are standard on the CPU motherboard.

**PDC:** Page Descriptor Cache; a cache that contains multiple TLB's, each holding a recently used translation.

**Physical:** Generally refers to system addresses that are decoded in order to select a memory location or device

**Reserved:** Addresses or bit fields that are not currently assigned within this architecture. Future implementations may assign these addresses or bit fields but should consult with the author first.

**Thread:** A *thread of execution* is a sequence of instructions executed either sequentially or with flow control managed by that execution. A thread has a program counter associated with it; each processor in a Sun-4M MP has a thread of execution associated with it. Threads are entered by an IU when it executes a trap. In software terms a thread generally has some process state associated with it; formal definitions can be found in SVR4 documentation.

**TLB:** Translation Lookaside Buffer; an entry in a PDC that contains a recently accessed translation. By caching translations an MMU can save accesses to page tables in memory.

**TSO:** Total Store Ordering; a slightly weak memory model defined in the SPARC Architecture Manual version 8.

**Virtual:** A virtual address is a 32-bit address issued by a SPARC IU or a DVMA master. A virtual address and a context number are translated by the MMU to a physical address.

### 3. Address spaces

#### 3.1 Virtual Address Space: Processor Cores

Virtual addresses are used only within the processor/cache module. Address space identifiers (ASI's) are interpreted by the cache interface logic. The modules contain memory management units (MMU's) that implement the SPARC Reference MMU, using TLB's and tables in main memory, which are used to translate virtual addresses into physical addresses when appropriate. This architecture supports up to four processor modules. Module #0 must be installed; the other modules may be installed in any order. All processor modules installed must be of the same type.

##### 3.1.1 Address Space Identifiers (ASI's)

The ASI code is interpreted by the processor and/or cache controller(s) on each module. No ASI information is accessible on the MBus. The ASI assignment is:

ASI	FUNCTION
0x00	Reserved
0x01	Reserved
0x02	Reserved
0x03	SRMMU flush/probe
0x04	Module Control/Status Registers
0x05	SRMMU Diagnostic, I-cache TLB
0x06	SRMMU Diagnostic, D-cache TLB (or I/D cache TLB if shared)
0x07	Unassigned
0x08	User Instruction
0x09	Supervisor Instruction
0x0A	User Data
0x0B	Supervisor Data
0x0C	I-cache Tag
0x0D	I-cache Data
0x0E	D-cache Tag (or I/D-cache tag, if appropriate)
0x0F	D-cache Data (or I/D-cache tag, if appropriate)
0x10	Flush I/D cache(s) by page
0x11	Flush I/D cache(s) by segment
0x12	Flush I/D cache(s) by region
0x13	Flush I/D cache(s) by context
0x14	Flush I/D cache(s) by user
0x15	Reserved
0x16	Reserved
0x17	Block Copy
0x18	Flush D cache by page
0x19	Flush D cache by segment
0x1A	Flush D cache by region
0x1B	Flush D cache by context
0x1C	Flush D cache by user
0x1D	Reserved
0x1E	Reserved
0x1F	Block Zero
0x20 – 0x2F	SRMMU Bypass (PA[35:32] = ASI[3:0], PA[31:0] = VA[31:0])
0x30 – 0x7F	Unassigned
0x80 – 0xFF	Reserved

} These 4 ASI codes indicate addresses that the MMU may translate

ASI spaces are accessed through use of the SPARC *lda*, *sta*, and *swapa* instructions. These instructions can only be issued in supervisor mode. In general making an access to a reserved ASI will result in a data access exception trap (exceptions to this are noted in the module appendices). ASI's 0x8 – 0xB are issued by SPARC IU's under normal data access and instruction fetch operations.

'Reserved' ASI's are reserved for future use by SPARC International. 'Unassigned' ASI codes may be used by system designers for design-specific functions.

For definitions of the ASI functions, see your module specifications (also Appendices B.I, B.II, B.III, B.IV, B.V). Not all ASI's are supported by all modules.

### 3.1.2 Translation Modes

Translation of virtual addresses to MBus physical addresses is done by the modules in the following modes:

ASI	BOOT-MODE	MMU EN	PA<35:00>	MBUS 'C' bit	NAME
0x8, 0x9	YES	X	PA<35:28> = 0xFF, PA<27:0> = VA<27:0>	0	Boot Ifetch
0x8, 0x9	NO	OFF	PA<35:32> = 0x0, PA<31:0> = VA<31:0>	0	Pass-thru
0x8, 0x9	NO	ON	PA<35:12> = PTE<31:08>, PA<11:0> = VA<11:0>	PTE<7>	Translate
0xA, 0xB	X	OFF	PA<35:32> = 0x0, PA<31:0> = VA<31:0>	0	Pass-thru
0xA, 0xB	X	ON	PA<35:12> = PTE<31:08>, PA<11:0> = VA<11:0>	PTE<7>	Translate
0x20-0x2F	X	X	PA<35:32> = ASI<3:0>, PA<31:0> = VA<31:0>	0	Bypass

When asserted the MBus 'C' bit indicates that this is a cacheable transaction. PTE/VA concatenation will differ if the PTE is not a level-3 PTE (4K page size). See the SRMMU specification in section 7.1 for details.

### 3.1.3 CPU Core Addresses (ASI = 0x4)

VA(31:00)	Description
0x00000000	Module Control Register
0x00000100	Context Table Pointer Register
0x00000200	Context Register
0x00000300	Synchronous Fault Status Register
0x00000400	Synchronous Fault Address Register
0x00000500	Asynchronous Fault Status Register
0x00000600	Asynchronous Fault Address Register
0x00000700	Reset Register
0x00000800 to 0xFFFFFFFF	Implementation Dependent: refer to the appropriate module specification.

For module-specific information, refer to the module specification. Current modules supported are Viking, Viking/MXCC, Ross-604 (level-1), and Ross-605 (level-2). See Appendices B.I, B.II, B.III, B.IV and B.V for the latest information on these modules. Not all of these modules will be use in real products. Actual module specifications, when available, will be more accurate.

## 3.2 Physical Address Space

### 3.2.1 Physical Address Space Allocation

The processor modules contain items that are accessed in ASI space with virtual addresses. The Address Space Identifiers (ASI's) and virtual addresses from the SPARC IU are interpreted and used solely on the modules. For ASI's 0x8, 0x9, 0xA, and 0xB the address is translated in to a 36-bit physical address (PA) by the SPARC Reference MMU (SRMMU) TLB in the module; the bypass ASI's 0x20-0x2F generate the PA without use of the MMU. DVMA virtual addresses are translated by the DVMA MMU (IOMMU). The 36-bit physical address space is further broken down into 16 32-bit address spaces identified by PA(35:32).

Reserved addresses are reserved for use in future implementations.

PA(35:32)	32-BIT SPACE	Optional?
0x0	Main Memory	No
0x1-0x8	Reserved	—
0x9	Memory-based Video ( <i>TBD</i> )	Yes
0xA	VME Master Port, User, 16-bit Maximum Data	Yes
0xB	VME Master Port, User, 32-bit Maximum Data	Yes
0xC	VME Master Port, Supervisor, 16-bit Maximum Data	Yes
0xD	VME Master Port, Supervisor, 32-bit Maximum Data	Yes
0xE	S-bus	No
0xF	Control Space	No

### 3.2.2 Control Space

PA(35:28)	Control Space (PA<35:32> = 0xF)
0xF0	Memory Control Space
0xF1-0xFC	Reserved
0xFD	VME/IOC Control Space
0xFE	S-Bus/IOMMU Control Space
0xFF	System Space

#### 3.2.2.1 On-Board Memory Control/Status Registers (32-bit access)

PA(35:00)	Memory-related Addresses	Section
0xF0000000	ECC Memory Enable Register	5.5.1
0xF0000004	Reserved	
0xF0000008	ECC Memory Fault Status Register	5.5.2
0xF000000C	Reserved	
0xF0000010	ECC Memory Fault Address Register 0	5.5.3.1
0xF0000014	ECC Memory Fault Address Register 1	5.5.3.2
0xF0000018	ECC Diagnostic Register	5.5.4
0xF0000020- 0xF0000FFF	Reserved	
0xF0001000- 0xF0001003	Diagnostic Message-Passing Registers <0:3>: byte access only	5.4.2
0xF0001004	Reserved	
0xF0FFFFFF		

3.2.2.2 System Space

PA(35:24)	System Space
0xFF0	EPROM
0xFF1	System Control Space
0xFF2	Reserved
0xFF3	Reserved
0xFF4	Reserved
0xFF5	Reserved
0xFF6	Reserved
0xFF7	Reserved
0xFF8	Control space for Module MBus master #8
0xFF9	Control space for Module MBus master #9
0xFFA	Control space for Module MBus master #A
0xFFB	Control space for Module MBus master #B
0xFFC	Control space for Module MBus master #C
0xFFD	Control space for Module MBus master #D
0xFFE	Control space for Module MBus master #E
0xFFF	Control space for Module MBus master #F

*For definitions of this space, see the appropriate module specification.*

Each Module MBus master represents the MBus interface for a processor cache. In the case of Harvard Architecture modules (split cache) the processor may require two master ID's, one for each cache. See the appropriate module specification to determine the allocation of addresses within these control spaces.

Most of the Module control space is implementation dependent; however, the highest 32-bit location in each slot (PA = 0xFFnFFFFFFC, where n = MID) is reserved for the MBus Port Address register (see section 4.7). The module ID is a 4-bit field provided by the MBus module connector. See the MID table in section 5.5.3 for definitions. The names used in this document (processor #<3:0>) are not the MID's, but rather are a Sun-4M convention.

3.2.2.2.1 System Control Space

PA(35:20)	System Control Space	Optional	Section
0xFF10	Keyboard/Mouse	No	9.1
0xFF11	Serial Ports	No	9.1
0xFF12	TOD/NVRAM	No	9.2
0xFF13	Timer/Counter and Counter Registers	Note †	5.3
0xFF14	Interrupt Registers	Note †	5.7
0xFF15	Audio/ISDN	Yes	9.4
0xFF16	Diagnostic LED's (write-only)	Yes	5.4.1
0xFF17	Floppy Controller	Yes	9.7
0xFF18	Auxiliary I/O registers	Yes	9.8
0xFF19	Reserved	-	
0xFF1A	Generic 8-bit Device	Yes	9.9
0xFF1B	Reserved	-	
0xFF1C	Reserved	-	
0xFF1D	Reserved	-	
0xFF1E	Reserved	-	
0xFF1F	System Status/Control Register	No	5.1.1

† Per-processor items are optional per implementation; one set is provided for each possible processor in that implementation.

3.2.2.2.2 Timer/Counter and Counter Registers

Address	Register	Type	Section
0xFF130N000**	Processor #N Limit Register or User Timer MSW	RW	5.3.2
0xFF130N004	Processor #N Counter Register or User Timer LSW	RW*	5.3.2
0xFF130N008	Processor #N Limit Register, doesn't reset Counter	W	5.3.2
0xFF130N00C	Processor #N User Timer Start/Stop Register	RW	5.3.4
0xFF1310000	System Limit Register (level 10 interrupt)	RW	5.3.2
0xFF1310004	System Counter Register	R	5.3.2
0xFF1310008	System Limit Register, doesn't reset Counter	W	5.3.2
0xFF131000C	reserved		
0xFF1310010	Timer Configuration Register	RW	5.3.1

\* Writable as User Timer LSW, read-only as Counter Register

\*\* 'N' is the processor number, starting at 0x0. One set of these registers is provided for each processor supported in an implementation.

3.2.2.2.3 Interrupt Registers

Address	Register	Type	Section
0xFF140N000*	Processor #N Interrupt Pending Register	R	5.7.1.1
0xFF140N004	Processor #N Clear_Pending Pseudo-Register	W	5.7.1.2
0xFF140N008	Processor #N Set_Soft_Int Pseudo-Register	W	5.7.1.3
0xFF140N00C – 0xFF140NFFF	Reserved	N/A	
0xFF1410000	System Interrupt Pending Register	R	5.7.3.1
0xFF1410004	Interrupt_Target_Mask Register	R	5.7.3.2
0xFF1410008	Interrupt_Target_Mask Clear Pseudo-Register	W	5.7.3.2
0xFF141000C	Interrupt_Target_Mask Set Pseudo-Register	W	5.7.3.2
0xFF1410010	Interrupt Target Register	RW	5.7.2
0xFF1410014 – 0xFF14FFFFF	Reserved	N/A	

\* 'N' is the processor number, starting at 0x0. One set of these registers is provided for each processor supported in an implementation.

3.2.2.3 VMEbus Control Space

PA(35:00)	VMEbus Control Space	Section
0xFD000000X	VMEbus Interrupt Vector Register	5.6.2
0xFD0000010 – 0xFDEFFFFFFF	Reserved	
0xFDF000000– 0xFDF007FFF	IOC Tags	8.3
0xFDF008000– 0xFDF00FFFF	IOC Data Diagnostic Access	8.4
0xFDF010000	VMEbus Interface Control Register	5.6.1
0xFDF010004	VMEbus Interface Asynchronous Error Address Register	5.6.3.2
0xFDF010008	VMEbus Interface Asynchronous Error Status Register	5.6.3.1
0xFDF01000C– 0xFDF01FFFF	Reserved	
0xFDF020000– 0xFDF027FFF	I/O Cache Flush	8.2
0xFDF028000– 0xFDFFFFFFFF	Reserved	

### 3.2.2.4 SBus/IOMMU Control Space

PA(35:00)	S-bus and IOMMU Control Space	Section
0xFE000000	IOMMU Control Register	5.8.1
0xFE000004	IOMMU Base Address Register	5.8.2
0xFE000008 –	Reserved	
0xFE000010	Reserved	
0xFE000014	Flush ALL TLB entries	7.2.4.1
0xFE000018	Address Flush Register	7.2.4.2
0xFE0000100	IOMMU Tags Diagnostic Access	7.2.6.2
0xFE0000200	IOMMU Translation Cache Diagnostic Access	7.2.6.1
0xFE0000300 –	Reserved	
0xFE0000FFF		
0xFE0001000	M-to-S Asynchronous Error Fault Status Register	5.2.1
0xFE0001004	M-to-S Asynchronous Error Fault Address Register	5.2.2
0xFE0001008	Arbiter Enable Register	5.1.2
0xFE000100C	Reserved	
0xFE00010ss	SBus Slot #N Configuration Register †, ss = (4 * N) + 0x10	5.9
0xFE0001060 –	Reserved	
0xFE0001FFF		
0xFE0002000	MID Register	5.4.3
0xFE0002004 –	Reserved	
0xFFFFFFFF		

† One SBus Slot Configuration Register is provided for each expansion slot supported. On-board devices are not given a Configuration Register.

There is no error checking on accesses to SBus control space (PA<35:28> = 0xFE). Writes to read-only registers are ignored (NOOP), reads of write-only addresses return undefined data, wrong-sized accesses have undetermined behaviour.

### 3.2.3 SBus Slot Addressing

PA(35:28)	S-Bus Slot	Section
0xEN †	S-Bus Slot #N	9.6, 10.4

† 'N' represents the SBus slot number. Implementations should number expansion slots starting at 0x0 and with increasing number, and on-board devices start at 0xF with decreasing number.

#### 3.2.3.1 SCSI/Ethernet Address Space

The on-board SBus devices will generally include a SCSI and Ethernet solution in SBus 'slot' 0xF. The details of this device are included in the system-specific 'A' appendix.

#### 3.2.4 Reserved Address Spaces

Accesses to addresses with PA<35:32> = 0x8 – 0x1 will result in a TIMEOUT acknowledge on the MBus. Accesses to other reserved spaces may not be detected, since addresses may not be fully decoded by all devices.

Access to reserved addresses within an address space allocated to a device may cause unspecified behaviour. Access to a reserved address space that is not allocated to a device will result in a timeout response. Unless otherwise noted, access to a non-installed device will result in a timeout response (some devices may require a write-read probe).



## 4. Processor Core Register Definitions (ASI = 0x4)

This chapter defines the core register assignments for a module. For module-specific details see the appropriate 'B' module appendix.

### 4.1 Module Control Register (MCR) (Address = 0x0000)

31	28	27	24	23	15	14	13	12	11	10	9	8	7	2	1	0
IMPL	VER	rsvd			BM	C	CP	CB	rsvd	CE	rsvd	NF	ME			

Field	Description	Type
IMPL	SRMMU implementation number	R
VER	SRMMU version number	R
BM	Boot Mode: 1 = Boot mode enabled	RW
C	Cacheable bit for second-level caches (MBus address phase); used when MMU is not enabled, or for transactions that do not use the MMU (i.e. table walks).	RW
CP	Cache Parameters. Meaning varies with IMPL and VER	R
CB	1 = Copy-back cache, 0 = write-through cache	RW †
CE	Cache Enable	RW
NF	No-Fault: When 1, Supervisor data access exceptions are not reported to the IU, but are captured in the fault status register.	RW
ME	MMU Enable	RW
rsvd	Reads as 0's; writing has no effect. Some modules have definitions for these bits; see the appropriate module appendix.	R

† In some implementations, only copy-back or write-through is supported, and this bit will be read-only.

### 4.2 Context Table Pointer Register (CTPR) (Address = 0x0100)

31	2	1	0
CTP<35:6>		rsvd	

Field	Description	Type
CTP	Context Table Pointer. This table-size-aligned physical address points to the context (CTX) table in memory, which is indexed by the CTX field of the context register. CTP will appear on MAD(35:6) when this level of table-walk is required.	RW
rsvd	Reserved. Reads as 0's, writing has no effect	R

### 4.3 Context Register (CTX) (Address = 0x0200)

31	N	N-1	0
rsvd		CTX	

Field	Description	Type
CTX	Context Number. This N-bit field contains the current context number. 2**N contexts are supported.	RW
rsvd	Reserved. Reads as 0's, writing has no effect	R

#### 4.4 Synchronous Fault Registers

The Synchronous Fault Status Register (SFSR) provides information on exceptions (faults) issued by the MMU as specified in the SPARC Reference MMU document. SRMMU is described in section 7.1.

Since the IU is pipelined, several faults may occur before a trap is taken. The faults are grouped into three classes; instruction access faults, data access faults, and translation table access faults. If an instruction access fault occurs before the status of a prior instruction access fault is read by the IU, the status of the latest fault is posted in the SFSR and the OW (overwrite) bit is set to indicate that status has been lost. If multiple data faults occur, the SFSR and SFAR contain the status related only to the fault that the CPU has trapped on. If a data fault overwrites status from an instruction access fault then the OW bit is cleared, since the fault status is represented correctly. Instruction access faults may not overwrite data access faults.

A translation table access fault occurs if an MMU access to the page tables causes an external system error. These faults will overwrite any previous data or instruction access fault, and will clear the OW bit. Data and instruction access faults may not overwrite a translation table access fault.

The Synchronous Fault Address Register (SFAR) captures the address of a data access or translation table access fault, and is valid when the FAV bit in the SFSR is set. The SFAR may not be valid for instruction access faults.

##### 4.4.1 Synchronous Fault Status Register (SFSR) (*Address = 0x0300*)

31	rsvd	13	12	11	10	9	8	7	5	4	2	1	0
		UC	TO	BE	L	AT	FT	FAV	OW				

Field	Description	Type
OW	Overwrite: multiple errors have occurred	R
FAV	Fault Address Valid: SFAR contains a fault address	R
FT	Fault Type: see table 4.4.3	R
AT	Access Type: see table 4.4.3	R
L	Page table level of fault, if fault was during a table-walk	R
BE	Bus Error response from MBus	R
TO	Timeout response from MBus or module interface	R
UC	Uncorrectable Error response from MBus	R
rsvd	reserved, read as '0'	R

This register is clear-on-read. Reading it also unlocks the Synchronous Fault Address Register.

If an instruction access fault occurs and OW is set, the software must probe the MMU and/or memory to determine what the exact fault was, using the saved PC as the fault address. The fault address register is not guaranteed correct for instruction access faults, and FAV will not be set if the SFAR is not valid.

##### 4.4.2 Synchronous Fault Address Register (SFAR) (*Address = 0x0400*)

31	SFA	1	0
----	-----	---	---

Field	Description	Type
SFA	Synchronous Fault Address: 32-bit virtual address of the fault	R

**4.4.3 Fault Type and Access Type Encoding**

FT	Fault Type	AT	Access Type
0	None	0	Load from User Data Space
1	Invalid Address Error	1	Load from Supervisor Data Space
2	Protection Error	2	Load/Execute from User Instruction Space
3	Privilege Violation Error	3	Load/Execute from Supervisor Instruction Space
4	Translation Error	4	Store to User Data Space
5	Access Bus Error	5	Store to Supervisor Data Space
6	Internal Error	6	Store to User Instruction Space
7	-reserved	7	Store to Supervisor Instruction Space

**4.4.4 L Encoding and Error Priorities (1 = Highest)**

L	MMU table level
0	Context Level
1	Region Level
2	Segment Level
3	Page Level

Priority	Fault Type
1	Internal Error
2	Translation Error
3	Invalid Address Error
4	Privilege Violation Error
5	Protection Error
6	Access Bus Error

**4.4.5 Translation Errors**

Invalid address, protection, and privilege violation errors depend on the Access Type field of the SFSR and the ACC field of the corresponding PTE. The errors are set as follows:

AT	FT Value								
	PTE[V] = 0		PTE[V] = 1, PTE[ACC] =						
		0	1	2	3	4	5	6	7
0	1					2		3	3
1	1					2			
2	1			2	2			2	3 3
3	1			2	2			2	
4	1			2		2		2	3 3
5	1			2		2		2	
6	1			2	2	2		2	3 3
7	1			2	2	2		2	2

A translation error is indicated if a bus error (UC, TO, or BE as defined in the SFSR description) occurs while the MMU is fetching an entry from a page table, if a PTP is found in a level-3 table, or if a PTE has ET = 3 (reserved). The L field indicates which level of page table was being accessed when the error occurred, and the UC, TO, or BE bit indicates what type of error was reported. Access Bus Error is set when any of these three error types are reported on a memory access that is not a table access by the MMU. Internal Error will be set if the MMU detects an internal inconsistency, and is considered a fatal error condition.

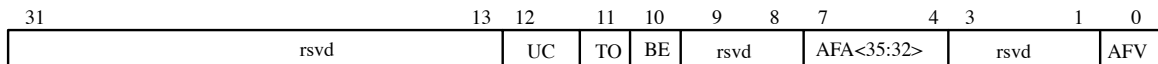
### 4.5 Asynchronous Fault Registers

The asynchronous fault registers capture fault information related to system bus errors that are reported on transactions that occur asynchronous to processor operations. Such transactions may include cache copy-backs, and stores that were accepted into a module write buffer. Asynchronous faults are reported to the processor (and to the rest of the system) via a level-15 broadcast interrupt, which is issued at the time that the AFV bit is set.

Clearing of the AFSR is controlled by reads of the AFAR. The AFSR should be read prior to the AFAR, and the AFAR should only be read if AFV (AFSR bit <0>) is asserted. This avoids a race condition between asynchronous faults being posted and accesses to the asynchronous fault registers.

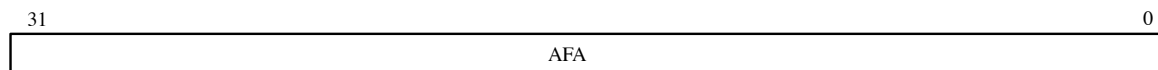
If asynchronous faults cannot occur in a particular implementation, that implementation can choose to not provide asynchronous fault registers. In that case accesses to these registers will provide either an error acknowledge or garbage data (the former is preferred) as specified in the module appendix. Asynchronous faults that occur on module write buffers further from the processor may be reported through implementation-specific error registers accessed through ASI 0x2.

#### 4.5.1 Asynchronous Fault Status Register (AFSR) (Address = 0x0500)



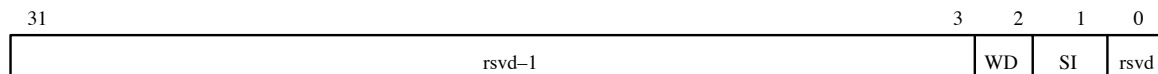
Field	Description	Type
AFV	Asynchronous Fault Occurred	R
AFA<35:32>	Asynchronous Fault Address bits PA<35:32>	R
BE	Bus Error response from MBus	R
TO	Timeout response from MBus	R
UC	Uncorrectable Error response from MBus	R
rsvd	reserved, read as '0'	R

#### 4.5.2 Asynchronous Fault Address Register (AFAR) (Address = 0x0600)



Field	Description	Type
AFA	Asynchronous Fault Address: PA<31:0> of the faulting address	R

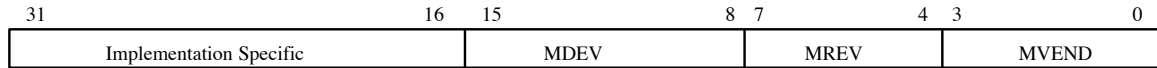
### 4.6 Reset Register (Address = 0x0700)



Field	Description	Type
WD	Watchdog Reset	R
SI	Software Internal Reset (not normally used)	RW
rsvd	Reserved, Write as '0'	
rsvd-1	Reserved, reads a '0', writing has no effect.	

See section 12: 'Resets' for a description of module and system reset functions.

**4.7 MBus Port Address Register** (See 3.2.2.2 for addressing)



Field	Description	Type
<b>MDEV</b>	<b>MBus device number: vendor specific device type</b>	<b>R</b>
<b>MREV</b>		<b>R</b>
<b>MVEND</b>		<b>R</b>
	<b>MBus vendor number: identifies the device vendor</b>	

Currently assigned vendor codes (for the MCR IMPL field and the Port Address MVEND field) are:

- 0x0 Fujitsu
- 0x1 Ross Technology/Cypress Semiconductor
- 0x2 (unassigned)
- 0x3 LSI Logic
- 0x4 Texas Instruments

## 5. System Register Definitions

Each register definition has the register access size specified in the section header. The size(s) stated is(are) the *only* valid access size(s) for each device; access of other size may be detected and reported, or may cause unspecified behaviour.

### 5.1 System Control Registers

#### 5.1.1 System Control/Status Register (32-bit access): (PA = 0xFF1F0000)

System Control/Status Register			
Bits	Name	Type	Meaning
D<0>	SW_RST	W	Generates the equivalent of a power-on reset Set on SW reset, clear on power-on or switch reset DIAG_MODE switch. 0 = normal mode, 1 = diag mode. Reads as '0' if this system does not support a diag switch
D<1>	SW_RST_STAT	R †	
D<2>	DIAG.SW	R	
D<3>	RST.SW	R †	Set on switch reset, clear on power-on or SW reset
D<31:4>	Reserved	R	Read as '0's

† write '0' to clear, writing '1' has no effect

#### 5.1.2 Arbiter Enable Register (32-bit access): (PA = 0xFE0001008)

Arbiter Enable (Diagnostic) Register				
Bits	Name	Type	Meaning	POR state
D<0>	Reserved	R	Read as '1'	1
D<1>	EN_P1_ARB	RW	Enables arbitration for MBus master 0x9 †	1
D<2>	EN_P2_ARB	RW	Enables arbitration for MBus master 0xA †	0
D<3>	EN_P3_ARB	RW	Enables arbitration for MBus master 0xB †	0
D<15:4>	Reserved	R	Read as '0'	0
D<16>	EN<0>	RW	Enables arbitration for SBus Slot 0	0
D<17>	EN<1>	RW	Enables arbitration for SBus Slot 1	0
D<18>	EN<2>	RW	Enables arbitration for SBus Slot 2	0
D<19>	EN<3>	RW	Enables arbitration for SBus Slot 3 ‡	0
D<20>	EN<F>	RW	Enables arbitration for <i>on-board</i> Sbus devices	0
D<30:21>	Reserved	R	Read as '0'	0
D<31>	SBW	RW	Enables S-to-M asynchronous writes. When '0' all such writes complete synchronously	0

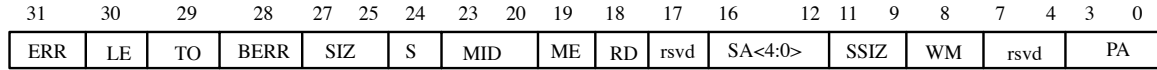
† When booting it is desirable to minimize the number of processors executing out of PROM in order to speed up the boot process. To meet this goal the processors connected to arbitration as processor\_2 (MID = 0xA) and processor\_3 (MID = 0xB) are disabled at system reset time. Processor\_0 (MID = 0x8) is enabled and is expected to act as the boot master, testing and initializing all system facilities prior to enabling arbitration for the other processors. At reset the arbitration for processor\_1 is enabled since some harvard-architecture modules will require one request/grant pair for data and another for instruction access. For configurations that are not split requesters, the system firmware should disable processor\_1 (MID = 0x9) arbitration early in the boot process.

‡ If an implementation plans more than 4 SBus expansion slots, please contact the author.

Arbitration can be arbitrarily enabled and disabled in a running system for diagnostic purposes. The disable function takes a small, indeterminate number of cycles to take effect due to the 'parking' nature of MBus arbitration; until a different master needs the bus a 'disabled' master may retain ownership.

**5.2 M-to-S Asynchronous Error Registers (32-bit access)**

**5.2.1 M-to-S Asynchronous Fault Status Register: (PA = 0xFE000100)**

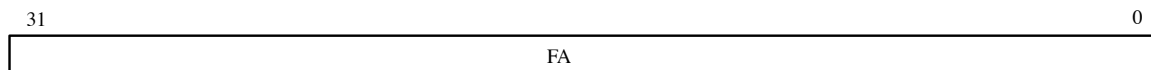


Field	Description	Type
<b>ERR</b>	<b>Summary error bit; LE or TO or BERR is asserted</b>	<b>R</b>
<b>LE</b>	<b>Late Error: SBus reported an error after the transaction was done</b>	<b>R</b>
<b>TO</b>	<b>Write access timed out in PA&lt;35:32&gt; = 0xE range</b>	<b>R</b>
<b>BERR</b>	<b>SBus write access received a BERR, or a write to the VME master port or E-bus received an error ack.</b>	<b>R</b>
<b>SIZ</b>	<b>Requested size of error transaction</b>	<b>R</b>
<b>S</b>	<b>Error access had the 'Supervisor' bit set in the MBus address phase</b>	<b>R</b>
<b>MID</b>	<b>Module ID number: indicates owner of the faulted cycle: See 5.5.3.4</b>	<b>R</b>
<b>ME</b>	<b>Multiple error: another error was detected after the error shown</b>	<b>R</b>
<b>RD</b>	<b>Direction of error access; may be a read for LE, all others are write</b>	<b>R</b>
<b>SA&lt;4:0&gt;</b>	<b>Actual address &lt;4:0&gt;; may differ from FA&lt;4:0&gt; if dynamic sizing or burst resizing when the error occurred</b>	<b>R</b>
<b>SSIZ</b>	<b>Actual size of error transaction (in case of dyn. sizing or burst resize)</b>	<b>R</b>
<b>WM</b>	<b>Interpret SSIZ as an SBus wide-mode access size (if supported)</b>	<b>R</b>
<b>PA</b>	<b>PA&lt;35:32&gt; of the fault address</b>	<b>R</b>
<b>rsvd</b>	<b>read as 0's, writing has no effect.</b>	<b>R</b>

SIZ<2:0>	Transaction (MBus encodings)
000	byte
001	half-word (2 bytes)
010	word (4 bytes)
011	double-word (8 bytes)
100	16-byte burst
101	32-byte burst
110	Reserved
111	Reserved

SSIZ<2:0>	Transaction (SBus encodings)	Transaction (Wide Mode)
000	word (4 bytes)	reserved
001	byte	reserved
010	half-word (2 bytes)	reserved
011	reserved	8-byte
100	16-byte burst	16-byte burst
101	32-byte burst	32-byte burst
110	Reserved	64-byte burst
111	8-byte burst	128-byte burst

**5.2.2 M-to-S Asynchronous Fault Address Register: (PA = 0xFE000104)**



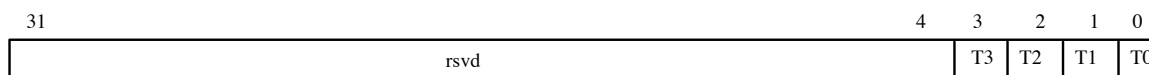
Field	Description	Type
<b>FA</b>	<b>Fault Address: PA&lt;31:0&gt; of the faulting access, as requested from the MBus. May be inexact (see 5.2.1)</b>	<b>R</b>

The M-to-S asynchronous fault register captures information related to errors that occur on MBus writes to the SBus, to VMEbus (second level write buffer error), or to SBus/IOMMU control registers. Such writes are write-buffered, so error reports will not be synchronous to the flow of instructions. Instead, if a write is scheduled in the MBus to SBus write buffer, and that write access receives either a timeout or error acknowledge, then the error information is frozen here. Errors will also be captured if the SBus reports a LATERR after the transaction. The ERR bit will be asserted, and a level-15 interrupt (source = M\_to\_S\_WRT\_BUF) is posted to the system. Subsequent errors are not captured, but the ME bit will be asserted. Any write to the FSR will clear the interrupt and the ERR bit, and will unlock both the FSR and the FAR.

Reads of the FSR will stall if the MBus to SBus write buffer is busy; this synchronization allows the kernel to determine if a write has completed. The MBus to SBus write buffer is used for write access from the MBus to SBus devices, E-bus devices, and the VMEbus master port.

### 5.3 Counter-Timer Registers and User Timers

#### 5.3.1 Timer Configuration Register (32-bit access): (Addresses in 3.2.2.2.2)



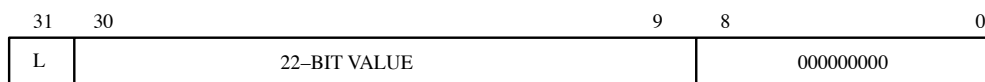
Field	Description	Type
T0	Timer/counter 0 configuration: 0 = Counter/Timer, 1 = User Timer	RW
T1	Timer/counter 1 configuration: 0 = Counter/Timer, 1 = User Timer	RW
T2	Timer/counter 2 configuration: 0 = Counter/Timer, 1 = User Timer	RW
T3	Timer/counter 3 configuration: 0 = Counter/Timer, 1 = User Timer	RW
rsvd	Reads as 0's, writing has no effect.	R

Each of the processor counter/timers (one per processor) can be configured to behave as a counter-timer with a level-14 interrupt, or to provide a real-time counter for high-resolution user performance analysis. In the first mode the timer is useful for OS kernel profiling. In the second mode the timer can be loaded upon each entry to user mode, and saved on each exit from user mode. By mapping the counter read-only for the user process, it provides 'virtual' time, a measure of the context run time, which can be used to measure code performance. It could also be loaded with a binary real time, which will then track precisely with the TOD.

Upon power-on reset all four sets are configured as counter-timers. The system level-10 timer/counter cannot be configured to act as a User Timer. The value of the User Timer and Count and Limit registers is unspecified after the corresponding configuration bit has been changed. It is required for software to initialize the counter after a mode change by writing to it in order to set the register value and to clear the limit bit. When the counter is programmed to be a User Timer the Counter/Timer function is disabled, and vice-versa; these functions share one counter, so no state is preserved across mode changes.

#### 5.3.2 Counter-Timer Registers (32-bit access): (Addresses in 3.2.2.2.2)

The counter-timer registers follow the structure of the counter-timers used in the Sun-4 architecture. Three addresses are associated with each counter; a COUNT register and a LIMIT register, and a load-limit-without-affecting-count pseudo-register. The counter and limit registers all look like



The counter increments at 500 nS intervals, with bit 9 as the LSB of the counter. When a counter reaches the value in the corresponding limit register, the Limit bit is set and the counter is set to 500 nS (i.e. 0x00000200). If the interrupt path for that counter is enabled, the presence of the L bit will assert the interrupt. The interrupt is cleared and the limit bit is reset when the corresponding Limit register is read. Reading the Count register will allow the value of the limit bit to be read without clearing it. The Counter register is read-only; when the timer is programmed to be a User Timer this register address is also writable.

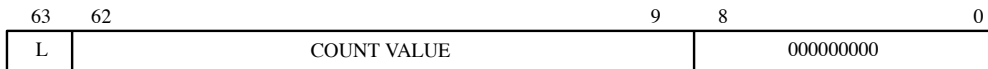
Writing to the Limit register sets the Count to 500 nS. Setting the limit register to '0' allows the counter to free-run. Since the timer always resets to a value of 500 nS after reaching maximum count, there is no match and no interrupts are generated. The Limit Register can also be written at a second address which loads the Limit Register without affecting the contents of the Counter Register; if the Count is already larger than the new Limit then the counter will count to its maximum value, then reset and count up to the Limit value before interrupting. The second address is



provided to allow for alarm-clock, as opposed to time-tick, usage of the timer/counter. The counter registers are not written to. Upon reset all limit registers are set to 0x00000000. Any write to the Limit register will clear that register's Limit bit independent of the data written.

The System Counter/Timer follows this structure, and provides an interrupt on level-10. Each of the processor counter/timers can be configured to be a counter/timer with a directed interrupt to the corresponding processor on level-14, or can be configured as a User Timer, through the Timer Configuration Register.

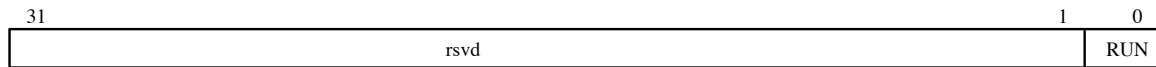
**5.3.3 User Timer Registers (64-bit access recommended; 32-bit access also allowed)**



When a timer is configured to be a User Timer, it should be accessed only as a 64-bit word. The counter increments at 500 nS in bit 9. The counter is read/write; it is recommended that the timer is mapped read-only for user-mode access. The 'L' bit is set any time the counter exceeds the maximum possible count value of 0x7FFFFFFFFFFFFE00, and is cleared on any write to the User Timer. **64-bit access on reads will ensure consistency between the high and low words.**

There is no interrupt associated with the User Timer function.

**5.3.4 User Timer Stop/Start Registers (32-bit access):** *(Addresses in 3.2.2.2.2)*



Bits	Name	Type	Meaning
D<0> D<31:1>	RUN RESERVED	RW R	When '1' the UT counts, when '0' the UT is frozen. Read as '0's

The User Timer Stop/Start Register associated with each User Timer is provided to allow fast trap handlers to stop the User Timer (UT) blindly during time-critical code without reading and saving the value. The UT must be restarted before re-entering user state. A software flag must be maintained to indicate if each User Timer is currently in use, so that the fast trap handler knows if it must restart the UT later. The contents of this register have no effect if the corresponding timer is configured as a counter/timer.

## 5.4 Diagnostic Support

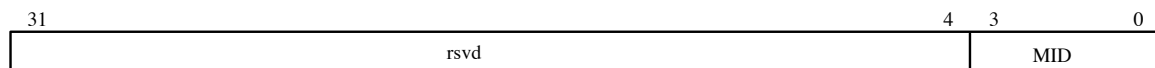
### 5.4.1 Diagnostic LED's (16-bit write-only access): (PA = 0xFF160000)

This register is a 16-bit write-only register. A '0' bit written to the register causes the corresponding LED to light up, a '1' will cause the LED to be dark. Upon power-up reset the diagnostic register is initialized to 0x0 causing all LEDs to light up. Depending upon the implementation fewer bits may be visible.

### 5.4.2 Diagnostic Message Registers: (PA = 0xF00001000 – 0xF00001003)

These four 8-bit registers are read/write, and accesses to them have no side effects. They are provided for diagnostic use. These registers are non-volatile across resets (except POR, which leaves them in a random state).

### 5.4.3 MID Register: (PA = 0xFE0002000)



Bits	Name	Type	Meaning
D<3:0>	MID	R	MBus Master ID of the current Bus master. Read as '0's
D<31:4>	RESERVED	R	

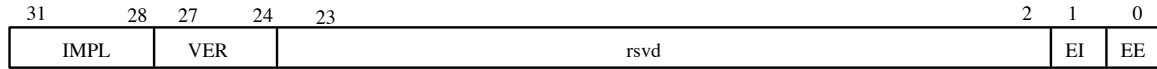
Some modules may not configure their MID in hardware; instead the system provides the MID information in this register so that the firmware can configure the MID. This *must* be done very early in the boot process, prior to processors accessing any system devices or registers with the exception of EPROM, this register, and the arbiter enable register. See 5.5.3.4 for the MID table.

**Note:** The first implementation of this register had a bug which renders the MID register nonfunctional. There are several methods which allow a processor to determine which MID it is. At configuration time the boot firmware can communicate with each processor in turn via use of the arbiter enable function, so that each processor can be uniquely identified. After this time, each processor can have a unique mapping to some page in memory where it can find out what MID it is; or information can be encoded in some internal module or IU register that can be used to identify a processor's MID. For example, each processor could have a unique value in its Trap Base Register (TBR) in the SPARC IU; all of the TBR's could map to the same physical page which contains the trap table, but each virtual address for the TBR can be unique and aliased. To determine the MID the IU can simply take the value in *%tbr*, mask, shift, and add to determine the MID.

This operation is required because there was no requirement that the modules provide a fixed location to read the MID from, so several modules exist that do not provide an MID in software readable form.

### 5.5 Memory Registers

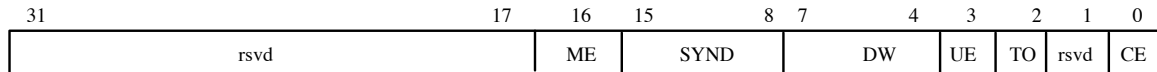
#### 5.5.1 ECC Memory Enable Register (32-bit access): (PA = 0xF0000000)



Field	Description	Type
EE	Enables ECC checking. Generation is always enabled. †	RW
EI	Enables Interrupt on correctable error. When '0' a CE will still be captured in the fault registers, but no interrupt will be generated. †	RW
IMPL	Identifies the Sun Implementation of this memory controller	R
VER	Version: Identifies the revision of this design	R
rsvd	Reads as 0's, writing has no effect.	R

† clears to 0 on power-on reset.

#### 5.5.2 ECC Memory Fault Status Register (EFSR) (32-bit access): (PA = 0xF0000008)



Field	Description	Type ‡
CE	Correctable Error during read or partial-write †	R
TO	Timeout on partial write access to expansion memory †	R
UE	Uncorrectable Error on partial write †	R
DW	Double-word within block that had the CE	R
SYND	Syndrome for Correctable Error	R
ME	Multiple Errors: an error was detected after the fault registers were frozen	R
rsvd	Reads as 0's, writing has no effect.	R

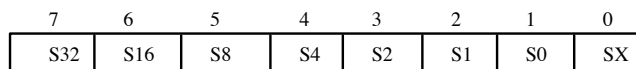
† clears to 0 on power-on reset.

‡ any write to the EFSR clears CE, UE and TO and un-freezes the EFAR.

Errors that are reported are (in order of priority, highest first) (1) partial write uncorrectable error, (2) correctable error, and (3) expansion memory timeout. If status for an error has been captured, and then a subsequent error of the same or lower priority is detected, the contents of the ECC Error Registers will not be over-written; instead the ME bit is set, and the appropriate error bit (UE, TO, or CE) is asserted. If multiple error bits are asserted then the one with the highest priority corresponds to the contents of the ECC Error Registers. The ME bit will be set any time that more than one error has been detected.

Access Errors (invalid size of access to control register, write to read-only register) will be reported synchronously to the MBus master as an MBus Bus Error.

If any of UE, TO, or CE is asserted then MEM\_ERR (level-15) is asserted. A write to the EFSR will clear error status and will release the interrupt. Reads of the EFSR will stall if the memory write buffer is busy; this synchronization allows the kernel to determine if a write has completed. The 8-bit Syndrome fields are defined as follows:



Sun-4M machines implement a SEC/S4ED Error Correction Code (ECC) based on a paper by Kaneda (IEEE Trans. on Computers, Aug84). This code provides correction of any single-bit error among 64 data bits and 8 check bits, as well as detection of errors in any two bits, or in any three or four bits within a nibble. The SYNDROME<7:0> field is interpreted as follows:

Syndrome Bits				S7	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
				S6	0	0	0	0	1	1	1	1	0	0	0	1	1	1	1
				S5	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1
				S4	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
S3	S2	S1	S0		C4	C5	D	C6	D	D	T	C7	D	D	T	D	T	T	Q
0	0	0	0	*															
0	0	0	1	C0	D	D	00	D	25	M	D	D	05	17	D	08	D	D	12
0	0	1	0	C1	D	D	01	D	29	36	D	D	M	21	D	13	D	D	09
0	0	1	1	D	32	33	D	42	D	D	M	47	D	D	M	D	T	T	D
0	1	0	0	C2	D	D	10	D	27	07	D	D	M	19	D	02	D	D	14
0	1	0	1	D	57	61	D	59	Q	D	M	63	D	Q	M	D	M	M	D
0	1	1	0	D	M	04	D	39	D	D	22	M	D	D	30	D	16	24	D
0	1	1	1	T	D	D	M	D	M	54	D	D	50	M	D	T	D	D	M
1	0	0	0	C3	D	D	15	D	31	M	D	D	38	23	D	03	D	D	11
1	0	0	1	D	37	M	D	M	D	D	18	06	D	D	26	D	20	28	D
1	0	1	0	D	49	53	D	51	Q	D	M	55	D	Q	M	D	M	M	D
1	0	1	1	T	D	D	M	D	M	62	D	D	58	M	D	T	D	D	M
1	1	0	0	D	40	45	D	34	D	D	T	35	D	D	T	D	M	M	D
1	1	0	1	T	D	D	T	D	M	48	D	D	52	M	D	M	D	D	M
1	1	1	0	T	D	D	T	D	M	56	D	D	60	M	D	M	D	D	M
1	1	1	1	Q	44	41	D	46	D	D	M	43	D	D	M	D	M	M	Q

\* – No error detected  
 Number – the bit number of the single-bit error  
 Note: All the Q-area is overlapped with D-area  
 D – double-bit error  
 T – triple-bit error  
 Q – quadruple-bit error  
 M – more than 4 bits errors

**5.5.3 ECC Memory Fault Address Registers (EFAR's) (32-bit access)**

These registers captures the MBus address of the transaction that caused the fault recorded in the ECC Memory Fault Status Register. Not all information is useful. VA is used only for the cache superset in virtual address cached systems. Any write to the ECC Memory Fault Status Register will unlock the address registers.

**5.5.3.1 ECC Memory Fault Address Register 0: (PA = 0xF0000010)**

31	28	27	26	22	21	14	13	12	11	10	8	7	4	3	0
MID		S	rsvd	VA		MBL	LOCK	C	SIZ	TYPE	PA<35:32>				

Field	Description	Type
PA	PA<35:0>: physical address of the faulting transaction	R
TYPE	TYPE<3:0>: Transaction type †	R
SIZ	SIZE<2:0>: Transaction size †	R
C	Address was mapped cacheable †	R
LOCK	Error occurred in an atomic cycle †	R
MBL	Boot Mode †	R
VA	VA<19:12>: superset bits for virtual address cache systems †	R
S	Access was supervisor mode	R
MID	Owner code	R

† These fields are required only in MBus-based implementations. They are captured during the address phase of faulting MBus transactions, and are provided for diagnostic purpose only.

**5.5.3.2 EFAR TYPE and SIZE Field Definitions**

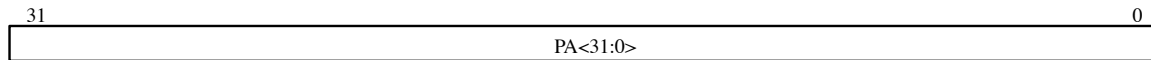
TYPE<3:0>	Transaction	SIZE<2:0>	Transaction
0000	Write	000	byte
0001	Read	001	half-word (2 bytes)
0010	Coherent Invalidate	010	word (4 bytes)
0011	Coherent Read	011	doubleword (8 bytes)
0100	Coherent Write and Invalidate	100	16-byte burst
0101	Coherent Read and Invalidate	101	32-byte burst
0110-1111	Reserved	110-111	Reserved

**5.5.3.3 MID Assignments**

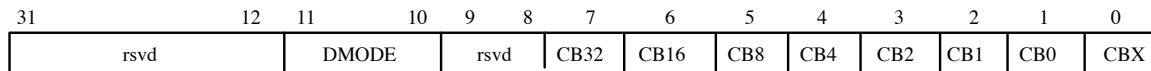
MID<3:0>	Owner	Non-Harvard	Harvard
0000	DVMA	-	-
0001-0111	Reserved	-	-
1000	MBus master 0x8	Processor 0	Processor 0 I-cache
1001	MBus master 0x9	Processor 1	Processor 0 D-cache
1010	MBus master 0xA	Processor 2	Processor 2 I-cache
1011	MBus master 0xB	Processor 3	Processor 2 D-cache
1100	MBus master 0xC	(future)	(future)
1101	MBus master 0xD	(future)	(future)
1110	MBus master 0xE	(future)	(future)
1111	MBus master 0xF	Processor 0 †	Processor 0 †

† A level-1 module will always issue MID = 0xF; it will be installed in slot '0'.

**5.5.3.4 ECC Memory Fault Address Register 1: (PA = 0xF0000014)**



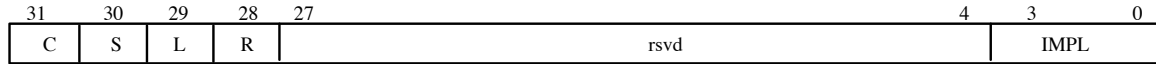
**5.5.4 ECC Diagnostic Register (32-bit access): (PA = 0xF0000018)**



Field	Description	Type
CBX	Diagnostic check bit CBX	RW
CB0	Diagnostic check bit CB0	
CB1	Diagnostic check bit CB1	
CB2	Diagnostic check bit CB2	
CB4	Diagnostic check bit CB4	
CB8	Diagnostic check bit CB8	
CB16	Diagnostic check bit CB16	
CB32	Diagnostic check bit CB32	
DMODE	Diagnostic mode: (resets to '00') 00: normal generate/detect/correct, diag disabled 01: diagnostic generate mode (CB bits substituted to memory) 10: diagnostic detect/correct mode (CB bits substituted to ECC check) 11: invalid	
rsvd	Reserved, no meaning	

**5.6 VMEbus Interface Registers**

**5.6.1 VMEbus Interface Control Register (32-bit access): (PA = 0xFDF01000)**

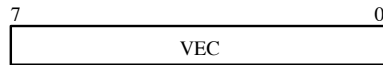


Field	Description	Type
C	I/OCache Enable: 1 = enabled, 0 = disabled. †	RW
S	SVME Enable: Enables the VME Slave port. †	RW
L	Loopback Enable: 1 = Diagnostic loopback enabled. †	RW
R	Reset VMEbus †	RW
IMPL	VME interface implementation number	R
rsvd	Reads as 0's, writing has no effect.	R

† clears to 0 on power-on reset.

Note: VME loopback is provided for diagnostic purposes only, and should not be used for normal operation of the system. Such accesses are very inefficient in time. Write buffer synchronization does not work for the loopback case; instead synchronization can be forced by issuing a VME loopback read that is not IOC-cacheable.

**5.6.2 VMEbus Interrupt Vector Register (8-bit access): (PA = 0xFD00000X)**

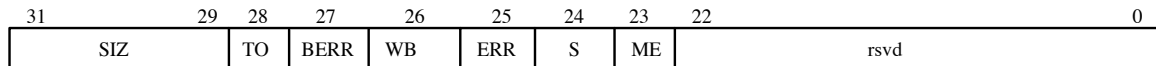


Field	Description	Type
VEC	VME Interrupt vector.	R

Reads of this register trigger a VMEbus 8-bit interrupt acknowledge cycle. Address bits A<3:1> indicate the acknowledge level. Address bit A<0> must = 1 for a valid cycle to occur.

**5.6.3 VMEbus Asynchronous Error Registers (32-bit access)**

**5.6.3.1 VMEbus Asynchronous Fault Status Register: (PA = 0xFDF010008)**



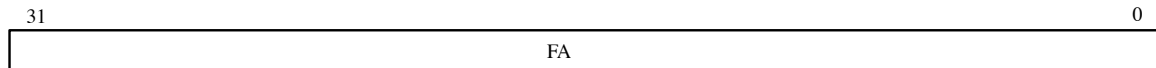
Field	Description	Type †
SIZ	Size of error transaction	R
TO	VME Master access timed out	R
BERR	VME Master access received a BERR	R
WB	IOC write-back error, or write error on non-IOC-cacheable store ‡	R
ERR	Summary bit; an error has occurred.	R
S	MVME error occurred on a supervisor-mapped space	R
ME	Multiple error: another error was detected after the error shown	R
rsvd	read as 0's, writing has no effect.	R

† Clear on write.

‡ SIZ = 32 indicates an IOC writeback error; other sizes indicate non-IOC-cacheable error. SVME writes are buffered, so the VME master was released before this error was reported.

SIZ(2:0)	Description
000	4 byte
001	1 byte
010	2 byte
011-100	reserved
101	32-byte
110-111	reserved

**5.6.3.2 VMEbus Asynchronous Fault Address Register:** (*PA = 0xFDF010004*)



Field	Description	Type
FA	Fault Address: 32-bit physical address of MVME fault or virtual address of IOC fault or non-IOC-cacheable SVME fault.	R

The error register will capture the first error to occur. Errors are not accumulated; subsequent errors are ignored and the ME bit is set to indicate that this has occurred. When an error is posted in these registers, the ERR bit is asserted, and a level-15 interrupt (source=VME\_control) is posted to the system. A write to the Fault Status Register will clear the interrupt, clear the error bit, and unfreeze the VFSR and VFAR so that subsequent errors will be captured.

VME Master reads will always receive a synchronous report of a timeout or BERR, via a memory exception. VME Master writes are scheduled through a write buffer, so any errors that occur are captured in the VFSR, and a level-15 interrupt (source=VME\_control) is posted. If the IOC receives an error from the IOMMU or the memory system, it will post the error to the VFSR with the WB bit asserted.

Synchronous errors will have their status posted in the module Synchronous Fault Status Register, so the IU will be able to determine if the error was a BERR or TO.

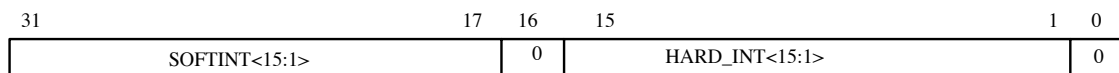
Reads of the VFSR will stall if the VMEbus write buffer is busy; this synchronization allows the kernel to determine if a write has completed.

**5.7 Interrupt Related Registers**

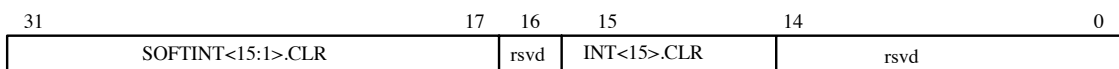
This section describes the contents and function of registers related to interrupts. Section 6 describes the multiprocessor interrupt distribution method.

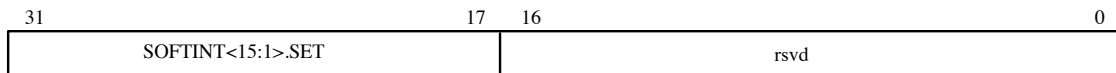
**5.7.1 Processor Interrupt Registers: 1 set per processor:** (*Addresses in 3.2.2.2.3*)

**5.7.1.1 Interrupt Pending Register: 1 per processor (32-bit access)(read only)**



**5.7.1.2 Clear-Pending Pseudo Register: 1 per processor (32-bit access)(write only)**

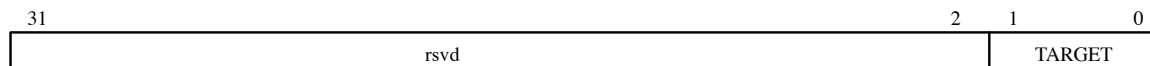


**5.7.1.3 Set-Soft-Int Pseudo Register: 1 per processor (32-bit access)(write only)**

The *interrupt pending* register shows soft interrupts and directed or undirected interrupts for this processor. Writes to the *set* pseudo-register with any bit set to 1 causes the corresponding bit in the pending register SOFTINT field to be set, which sends a directed interrupt at that level to the corresponding processor. Writes to the *clear* pseudo-register with any bit set to 1 causes the corresponding SOFTINT or level-15 broadcast interrupt bit to be cleared, acknowledging that interrupt. Shared message areas in memory must be used for the interrupt recipient to determine which processor sent the interrupt.

The set/clear mechanism allows for single-operation atomic access to this register, eliminating the need for *mutex* locks around such accesses. All pending interrupt bits are cleared upon system RESET.

HARD\_INT<13:1> will only be active if this processor is specified as the Current Interrupt Target in the Interrupt Target Register; otherwise they read as '0'. These bits are never 'set'; rather, they reflect the current status of interrupts on the corresponding level if this processor is enabled to see them (via the Interrupt Target Register). HARD\_INT<14> corresponds to this processor's counter/timer.

**5.7.2 Interrupt Target Register: 1 per system (32-bit access): (PA = 0xFF1410010)**

Field	Description	Type
TARGET	Current processor target for the undirected interrupts	RW
rsvd	read as 0's, writing has no effect.	R

The TARGET code indicates which processor receives the undirected interrupts. In order to prevent spurious interrupts, the current target should have traps disabled when it is relinquishing the Interrupt Target Register. Alternatively, if a processor wants to take ownership of interrupts, i.e. become the *Current Interrupt Target* (acquire CIT) there is a window of time where a new interrupt could be asserted, the CIT could change, and the old CIT will respond to the interrupt, only to discover that it has no interrupts showing in its Interrupt Pending Register. There is no way in hardware to prevent this race; for this reason a relinquish algorithm is preferred. A processor receiving a spurious interrupt should check the Interrupt Target Register to verify that this race was the cause.

**5.7.3 System Interrupt Pending Registers: 1 set per system (32-bit access)**

In the Sun-4M architecture, all interrupts sources are visible in a single 32-bit register called the System Interrupt Pending Register. When a particular interrupt is asserted, a '1' is visible in the corresponding bit in this register. Depending on the current value of the Interrupt Target Mask and the Interrupt Target registers, the assertion of these interrupts will cause the Interrupt Target to receive an interrupt at the appropriate level. The level-15 broadcast interrupts are also visible in the System Interrupt Pending Register.

Bits in the System Interrupt Pending Register are never 'set' or 'cleared'; rather, they reflect the status of the corresponding interrupt source. When the (level-sensitive) interrupt is released at the source by the interrupt handler, the assertion of the interrupt will be released and the bit visible in the System Interrupt Pending Register will reflect that change. There is an indeterminate latency between the write to release an interrupt and the actual deassertion of that interrupt. If necessary this can be dealt with either with delay loops or by polling of the appropriate interrupt bit in the SIPR.



Associated with the System Interrupt Pending Register is the Interrupt Target Mask Register. This register has three addresses, one for reading, and one each for setting and clearing individual bits (in the same manner as the processor soft\_int mechanism). When a bit in the Interrupt Target Mask Register is asserted, the current Interrupt Target will not see any effect from the assertion of the corresponding interrupt.

This mask is provided to support the allocation of interrupts to different processors. When the Current Interrupt Target (CIT) receives an interrupt, it can determine the source by examining the System Interrupt Pending Register. If it wants to schedule the service of this interrupt on another processor, it can leave a message in memory, SET the associated mask bit, and send a directed interrupt to the selected servicer. The masking function allows the CIT to continue without receiving another trap due to the presence of that particular interrupt source. (Note that interrupts are level-sensitive). When the selected servicer has serviced the interrupt, it will CLEAR the mask bit so that subsequent interrupts from that source will reach the CIT.

The level-15 interrupt sources are also visible in the System Interrupt Pending Register, and are maskable with the Interrupt Target Mask Register. While these interrupts are considered 'non-maskable' within the SPARC IU, a mask capability is provided to allow the boot firmware to establish a basic environment before receiving any level-15 interrupts, which are non-maskable within SPARC. A mask-all bit is provided to allow disabling of all external interrupts during change of the CIT. All mask bits are SET upon system reset.

Note that the mask for level-15 interrupts will prevent broadcast of those interrupt sources; that is, a level-15 source will only cause a broadcast interrupt to occur if that source's mask bit was not set at the time of the interrupt event.

**5.7.3.1 System Interrupt Pending Register (read only) (32-bit access): (PA = 0xFF141000)**

31	30	29	28	27	26	23	22	21	20	19	18	17	16	15	14	13	7	6	0
rsvd	ME	I	M	V	rsvd	FL	MI	VI	T	SC	A	E	S	K	SBUS	VME			

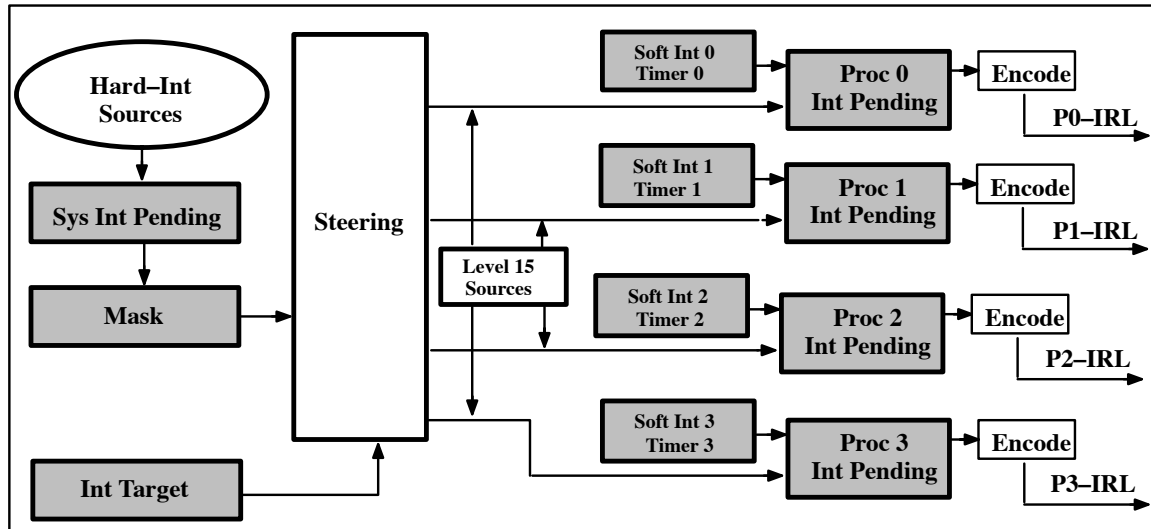
Field	Description; applies to both SIPR and ITMR
VME	VME interrupts <7:1>
SBUS	SBus interrupts <7:1>
K	Keyboard/mouse SCC
S	Serial ports SCC
E	On-board Ethernet
A	Audio/ISDN
SC	On-board SCSI
T	Level-10 Timer/Counter
VI	On-board Video Interrupt
MI	Module Interrupt (future non-processor modules)
FL	Floppy disk interrupt
MA	Mask All interrupts. Does not affect the state of other mask bits <i>Broadcast Interrupt Sources (level 15)</i>
V	VME asynchronous error (write buffer or IOC write-back err)
M	ECC Memory Error
I	M-to-S write buffer error
ME	Module Error (any module)
rsvd	read as 0's, writing has no effect.

**5.7.3.2 Interrupt Target Mask Register (read only), mask Set and Clear (write only) (32-bit access)**

(PA = 0xFF1410004, 0xFF141000C, 0xFF1410008)

31	30	29	28	27	26	23	22	21	20	19	18	17	16	15	14	13	7	6	0
MA	ME	I	M	V	rsvd	FL	MI	VI	T	SC	A	E	S	K	SBUS	VME			

**5.7.4 Interrupt Flow Diagram**



**5.8 IOMMU Registers**

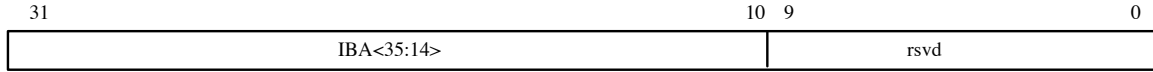
Functionality and management of the IOMMU is described in section 7.2.

**5.8.1 IOMMU Control Register: (PA = 0xFE000000)**

31	28	27	24	23	5	4	2	1	0
IMPL	VER			rsvd		RANGE	DE	ME	

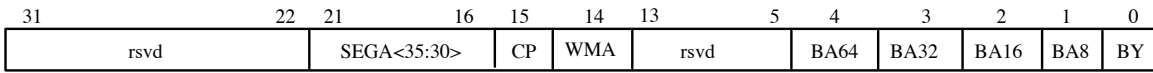
Field	Description	Type
IMPL	Implementation number of IOMMU	R
VER	Version number of IOMMU	R
ME	IOMMU Enable: 1 = translation is enabled. 0 = pass-through mode; PA<31:0> = VA<31:0>, PA<35:32> = 0000 Cleared to 0 on reset.	RW
DE	Diagnostic Enable; when 1, allows direct access to tags and TLB. If DE = 1 and ME = 1, care must be exercised to not introduce inconsistencies in the LRU queue or tags. When '0' diagnostic accesses are treated as NOOPs.	RW
RANGE	IOMMU translation range = 16 MB * 2**<RANGE>. Resets to 000.	RW
rsvd	Reads as 0's, writing has no effect.	R

**5.8.2 IOMMU Base Address Register: (PA = 0xFE000004)**



Field	Description	Type
IBA	<b>IOMMU Base Address:</b> when IOMMU is enabled and the access translation misses on the TLB, IBA is used as the base address for the (<RANGE/1024>)byte-aligned IOMMU Page Table.	<b>RW</b>
rsvd	Reads as 0's; writing has no effect	<b>R</b>

**5.9 SBus Slot Configuration Registers (One per SBus expansion slot): (Addresses in 3.2.2.3)**



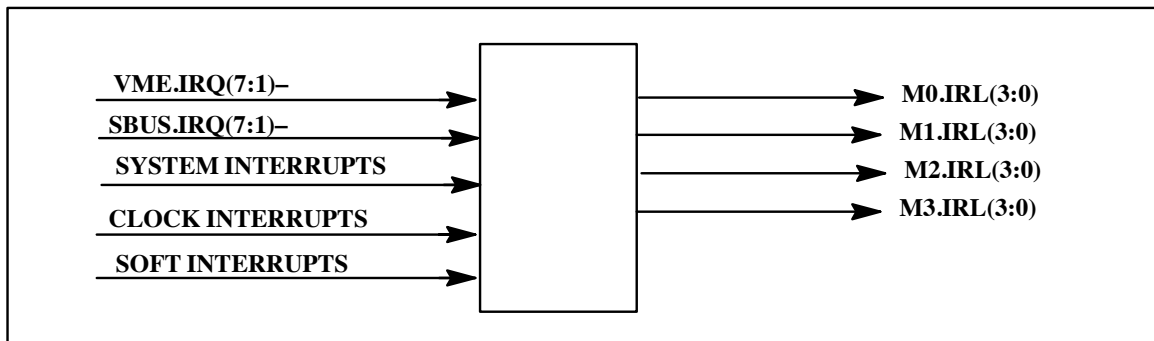
Field	Description	Type
SEGA<35:30>	<b>Segment Address:</b> provides PA<35:30> when IOMMU bypass is used	<b>RW</b> †
CP	<b>Cacheable bit:</b> used in physically cached systems to determine if bypass access should be treated as coherent on MBus.	<b>RW</b>
WMA	Enables wide-mode accesses if supported, else writes are ignored	<b>RW</b>
BA64	Slave supports 64-byte bursts	<b>RW</b>
BA32	Slave supports 32-byte bursts	<b>RW</b>
BA16	Slave supports 16-byte bursts	<b>RW</b>
BA8	Slave supports 8-byte bursts	<b>RW</b>
BY	Bypass Enabled	<b>RW</b>
rsvd	Reads as 0's; writing has no effect	<b>R</b>

† In some implementation, some of SEGA may be hardwired to '0'

Note that the on-board devices do not have Slot Configuration Registers associated with them. These are provided only for expansion slots.

## 6. Interrupts

There are 15 levels of external interrupts that can be accepted by the modules. These are encoded in a 4-bit code IRL(3:0), with the code of '0' indicating that no interrupts are pending. In an MP environment the capability is needed to steer individual interrupts to any of the four processors. It is also necessary to have the ability to post interrupts via software ('soft' interrupts) to any of the processors at various levels. With both a VMEbus and an SBus supported in the system, interrupt levels may be shared by many devices in different environments and the O/S should have the capability to select one or more source to dispatch to a processor (or to an interrupt handling thread), and to isolate sources so that different drivers can migrate easily from processor to processor. The hardware must be flexible so that software can choose the best algorithms for servicing the interrupts.



Section 6.1 deals with definitions of interrupts in an MP environment. Sun-4M-specific details follow in section 6.2

### 6.1 Interrupt Definitions

#### 6.1.1 Interrupt Types

There are three types of interrupts in multiple processor systems. A *directed interrupt* is an interrupt that is always posted to the same target processor. Directed interrupts can come from specific devices, from error conditions, and from *soft\_interrupts* posted from any processor. An *undirected interrupt* comes from a system device, and the interrupt steering logic selects a target processor based upon some distribution scheme. A *broadcast interrupt* is an interrupt that is sent to all processors in the system.

Interrupts in SPARC are defined in 15 interrupt priority levels, with level-15 as the highest priority. The interrupts pending to a particular processor are priority encoded and sent to the SPARC processor as a 4-bit code IRL(3:0). A code of '0' indicates that there are no interrupts pending. Asynchronous interrupt sources should be properly synchronized prior to encoding to prevent spurious codes.

#### 6.1.2 Processor to Processor Interrupts

Associated with each processor is a register set that posts soft interrupts at any level. The register can have any bit asserted by writing to the associated *set* register with that bit asserted, and can have any bit cleared by writing to the associated *clear* register with that bit asserted. The deasserted bits in the data written have no effect. This mechanism eliminates the need for *mutex* locks around accesses to these registers.

There is no information about the source of the *soft\_interrupt* in hardware. The kernel must establish message areas in shared memory that can be polled by the recipient of the *soft\_interrupt* to find the pending source(s). Any queuing of multiple soft interrupts must also be treated in the message areas.

### 6.1.3 Directed Interrupts

Certain interrupts must always be sent to a particular processor. For example, the level-14 high-resolution timer interrupts are directed; one timer/counter is dedicated to each processor. Note that the level-10 time-tick interrupt is undirected, and there is one level-10 timer/counter for the entire system.

Some device drivers are not reentrant, and therefore some interrupts may not be distributed to multiple processors.

### 6.1.4 Undirected Interrupts

Undirected interrupts come from system devices. The servicing of these interrupts can be shared among the different processors in the system.

For a small MP system, i.e. 4 or fewer processors, the expected interrupt traffic is small enough that a single processor can handle identifying sources and scheduling different processors to service those interrupts, via memory descriptors and directed interrupts. In a large MP system, the allocation may be handled in hardware. The hardware/software distribution criterion is that, if a single processor receives all undirected interrupts and schedules them for service on the different processors in the system, the time spent should be small enough that a process running on that processor will not be significantly slower than a process running on any other processor in the system. A maximum 'scheduling' load of 5% is recommended.

In order to support more flexible allocation algorithms, the 'current target' is programmable; that is, one of the processors in the system can be selected to receive all undirected interrupts.

### 6.1.5 Broadcast Interrupts

A broadcast interrupt is an interrupt that is posted to all processors in the system, including the processor that initiates the posting (if the interrupt is from a processor). A broadcast interrupt can be issued at any interrupt level. Implementation of broadcast is system dependent; in a small-MP system, it may be implemented with a series of processor-to-processor interrupts, while in larger systems it may be issued as a single message. The effect of a broadcast interrupt is to set the same level *soft\_interrupt* for all processors. The issue of a broadcast interrupts should be done in a single kernel routine that hides the differences in implementation. The acknowledge of broadcast interrupts must be handled in the shared-memory message area used for *soft\_interrupts*.

## 6.2 Sun-4M Interrupt Structure

### 6.2.1 Interrupt Distribution

Sun-4M is a small MP system, consisting of 1 to 4 processors. Each processor will receive directed interrupts for level-14 profiling. Each processor has a facility to receive a directed interrupt on any level, which is asserted by writes to a register associated with that processor. All undirected system interrupts will go to the processor indicated by the Interrupt Target Register, and that processor (referred to hereafter as the Current Interrupt Target, or CIT) can schedule the interrupt to any processor via the directed interrupt mechanism and shared memory communication, or can service the interrupt. All hard level-15 interrupts will be broadcast to all processors; the assertion of any level-15 source will set the level-15 bit in each Processor Interrupt Pending register, and each processor can acknowledge by clearing its own copy of that bit. Only one processor may service the interrupt source, but each processor can clear its own pending bit.

A read-only register shows the interrupts that are asserted at any time. When the Current Interrupt Target processor has scheduled an interrupt to a processor for service, it will write a mask bit in a corresponding register so that the presence of that interrupt will no longer cause the CIT to trap for that interrupt source. When the processor that is

scheduled has serviced the interrupt, it must also de-assert that mask bit so subsequent interrupts from that source will again go to the CIT.

The registers associated with interrupts are (1) the Processor Interrupt Registers, (2) the System Interrupt Registers, (3) the Timer/counter Registers, and (4) the Interrupt Target Register.

### 6.3 Interrupt Level Assignment

LEVEL	SOURCES
1	SOFTINT.1
2	SOFTINT.2, VMEbus L1, Sbus L1
3	SOFTINT.3, VMEbus L2, Sbus L2
4	SOFTINT.4, on-board SCSI
5	SOFTINT.5, VMEbus L3, Sbus L3
6	SOFTINT.6, on-board Ethernet
7	SOFTINT.7, VMEbus L4, Sbus L4
8	SOFTINT.8, on-board video
9	SOFTINT.9, VMEbus L5, Sbus L5, Module Interrupt (non-IU)
10	SOFTINT.10, System Counter/Timer
11	SOFTINT.11, VMEbus L6, Sbus L6, Floppy (PIO)
12	SOFTINT.12, Keyboard/Mouse, Serial Ports
13	SOFTINT.13, VMEbus L7, Sbus L7, ISDN Audio (PIO)
14	SOFTINT.14, Per-processor counter/timer
15	SOFTINT.15, Asynch. Errors (broadcast)

Note 1: Soft Interrupts are per processor

Note 2: SBus mapping to SPARC IRL in Sun-4M is different from Sun-4c

### 6.4 Programming Notes on Error Reporting

Errors that happen during a read operation are always reported back to the IU as a *data access* or *instruction access exception* (or *error*) trap. Status about such an error is captured in that module's Synchronous Fault Status Register and Synchronous Fault Address Register.

Errors that occur during a write may be reported either synchronously in the same way that read errors are reported, or asynchronously via a level-15 broadcast interrupt. Write errors are reported asynchronously when the write has been accepted by a write buffer, which has then released the processor to do useful work concurrent with the completion of the write. Such write buffers exist in the M-to-S interface, the VME interface, and the memory controller. Each write buffer has an associated Asynchronous Fault Register set. Detection of an error causes the fault address and status to be captured, and a level-15 broadcast interrupt to be posted.

A broadcast level-15 interrupt will interrupt all processors in the system, and will set the HARDINT.15 bit in each processor's Interrupt Pending Register. Each processor can remove its own level-15 interrupt by clearing that bit in its own IPR; the kernel must provide a mechanism so that one and only one processor services the source of the error interrupt. Sources are visible in the System Interrupt Pending Register, and are serviced in the appropriate system AFSR. Level-15 sources can be masked in the Interrupt Target Mask Register.

Modules may also contain a write buffer used for certain operations such as cache write-back. Again, these operations occur asynchronous to the flow of instructions in the IU. If an error is reported back to the module for a write operation that uses this write buffer (i.e. does not stall the processor) then that error will be captured in the module Asynchronous Fault Status/Address Register pair, and the module will post a Module Error to the system; that error in turn will be sent as a broadcast level-15 interrupt to all processors in the system. This is done to bring the MP environment to a simple state while the error is sorted out. A watchdog reset is sent as a level-15 broadcast to all processors, for the same reason. Note that accesses posted in a module write buffer have already been successfully translated; translation errors are always posted in the module's Synchronous Fault Status and Address Registers as defined in chapter 4.

The sequence that is recommended when a level-15 interrupt is received is the following:

(1) Read the processor's Interrupt Pending Register to determine if the interrupt is a SOFT\_INT or a HARD\_INT. If it is a SOFT\_INT then it was sent by a processor, and there should be information in memory as to why the directed interrupt was sent. Otherwise, it is a hardware broadcast interrupt.

(2) write to the processor's Interrupt Pending Clear pseudo-register with data = 0x00008000 to clear the HARDINT.15 bit.

(3) try to acquire the L15\_SERVICE\_LOCK (i.e. a *mutex* around ownership of L15 service). If fail, goto (5)

(4) as owner of the L15\_SERVICE\_LOCK, read the System Interrupt Pending Register to determine the source(s) of the level-15 broadcast interrupt;

-If memory then read the ECC Fault Status Registers

-If M-to-S then read the M-to-S Asynchronous Fault Status Registers

-If VME then read the VME Asynchronous Fault Status Registers

-If Module Error

-If a non-processor module is installed, may require special handling here.

-otherwise, release the L15\_SERVICE\_LOCK and goto (5)

(5) Each processor will check its own Asynchronous Fault Status Register in ASI = 0x4 space, take appropriate action; also check the Module Reset Register in ASI = 0x4 space to see if the Module Error was reported due to a watchdog reset on this module. Note that a watchdog will reset the one processor, which will go through a watchdog restart; part of that procedure might be to check and service the processor's HARDINT.15 that remains due to the watchdog.

(6) check the HARDINT.15 bit again in case another broadcast has occurred. If so, goto (1).

## 7. Memory Management and Cache Coherence

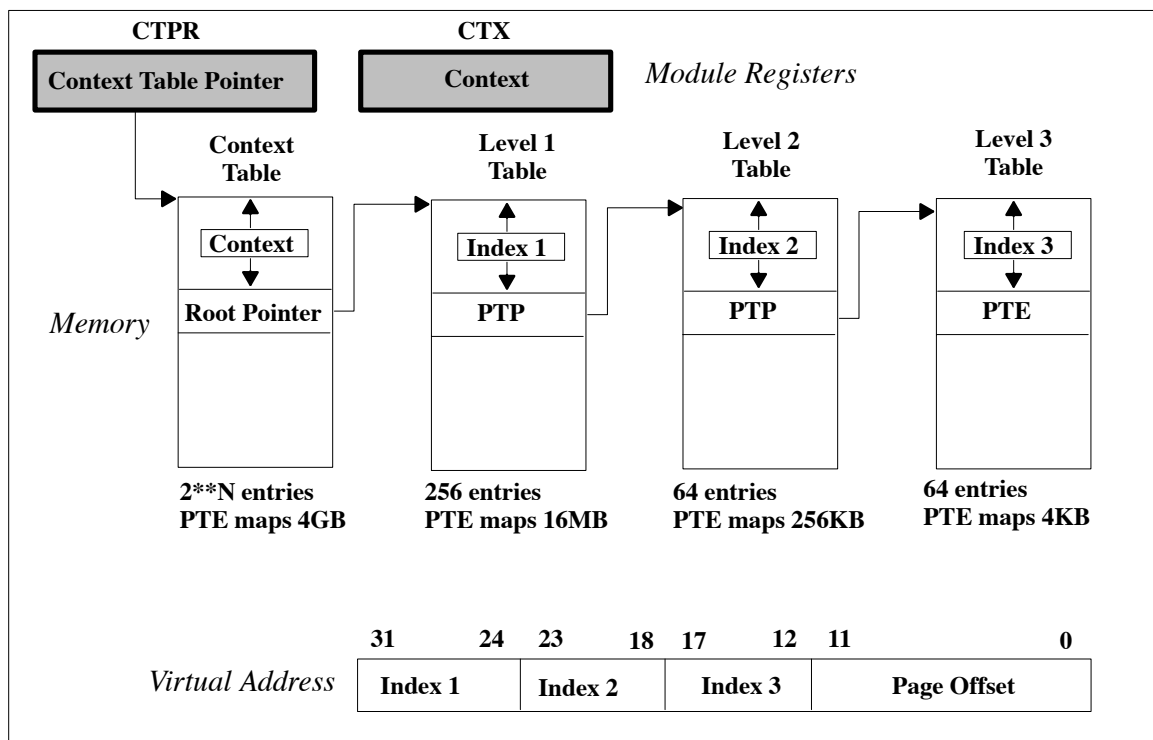
### 7.1 SPARC Reference MMU

Processor modules will implement a SPARC Reference MMU (SRMMU) local to each processor. In the case of virtual Harvard Architecture modules, there may be two SRMMU controllers on a module, one for each cache. For further information about the SRMMU, see ref. [1].

Main memory is allowed to be mapped cacheable. *No other element may be marked cacheable.* The kernel must enforce this rule.

In systems with virtual-address caches (i.e. Ross 605) the page tables are not cacheable; that is, the pages in which the SRMMU tables are stored must be accessed non-cacheable by the memory management software, since the MMU's access them without translation on table walks (obvious, but important). In systems with physical address caches (i.e. Viking) it is legal for the processors to treat the pages containing the SRMMU tables as cacheable; in fact, this is recommended for efficiency in Viking-based systems.

#### 7.1.1 Overview of SRMMU



SPARC Reference MMU (SRMMU) provides translations from a 32-bit virtual address to a 36-bit physical address through the use of tables in main memory. Each MMU controller contains a set of Translation Look-aside Buffers (TLB's) to keep recently used translations cached close to the processor. The set of TLB's is referred to as a Page Descriptor Cache, or PDC. A PDC can contain both page table entries and page table pointers.

When a new translation is required the MMU controller will do a *table walk* to find it. As shown in the above diagram a series of tables is accessed. At any level the controller will find either a Page Table Pointer (PTP) or Page Table Entry (PTE). A PTP contains the physical address of the next table to access, while a PTE contains the actual



translation information. Depending on the level at which a PTE is found, the page size could be 4KB, 256KB, 16MB, or 4GB. The offset within page is not translated; for example in a 4KB page the physical address bits <11:0> equal the virtual address bits <11:0>.

This multiple-level mapping provides for efficient translation of sparse address spaces. It is unlikely that a full set of tables is required.

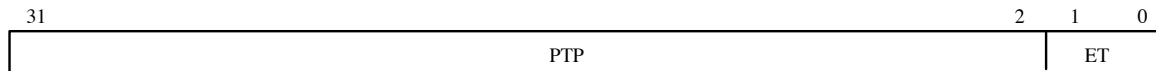
**7.1.2 Registers Associated With SRMMU**

Most of the module registers defined in chapter 4 are defined as part of the SPARC Reference MMU. Most of the Module Control Register (MCR) definition comes from SRMMU. The Context Table Pointer Register (CTPR) and Context Register (CTX) provide the 'root pointer' for a context's level-1 table. The Synchronous Fault Status Register (SFSR) and Synchronous Fault Address Register (SFAR) are also part of the SRMMU, while the Asynchronous Fault Status and Address Registers (AFSR and AFAR) are not.

Chapter 4 contains definitions of all of the fields in these registers, and a description of the behaviour of the synchronous fault registers.

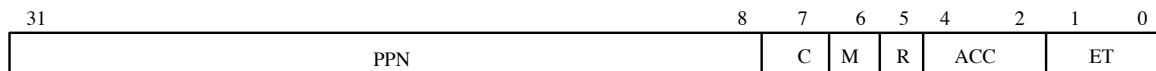
**7.1.3 SRMMU PTE's and PTP's**

**7.1.3.1 SRMMU PTP's (Page Table Pointers)**



Field	Description
PTP	<b>Page Table Pointer:</b> provides PA<35:6> of the base address of the next-level page table during a table walk. The page table pointed to must be aligned on a boundary equal to the size of the table pointed to, as defined in the diagram in section 7.1.1.
ET	<b>Entry Type:</b> for a PTP, ET = 1. ET = 0 means Invalid, ET = 3 is reserved

**7.1.3.2 SRMMU PTE's (Page Table Entries)**



Field	Description
PPN	<b>Physical Page Number:</b> this field provides PA<35:12> for a translated address; depending on the page size, bits <35:N+1> are concatenated with VA<N:0> to provide the entire translated address.
C	<b>Cacheable:</b> When '1', indicates that this page is cacheable by a data and/or instruction cache
M	<b>Modified:</b> This bit gets set to '1' by the MMU when the page is accessed for writing.
R	<b>Referenced:</b> This bit gets set to '1' by the MMU when the page is accessed.
ACC	<b>Access Permissions:</b> defined in the table in 7.1.3.3.
ET	<b>Entry Type:</b> for a PTE, ET = 2.

**7.1.3.3 Access Permissions**

ACC	Permissions	
	User	Supervisor
0	Read Only	Read Only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Read/Write/Execute	Read/Write/Execute
4	Execute Only	Execute Only
5	Read Only	Read/Write
6	No Access	Read/Execute
7	No Access	Read/Write/Execute

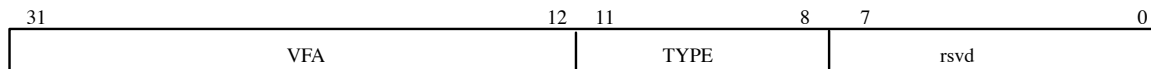
**7.1.3.4 'R' and 'M' Bits**

A successful translation of any kind results in the Referenced bit in the PTE being checked. If it is '0' then the MMU will set it to '1' in both the PTE and in the cached copy of the PTE in the PDC. A successful translation of a write operation results in the Modified bit being checked. If it is a '0', the MMU will set it to '1' in both the PTE and the PDC.

The SRMMU specification states that the Modified bit gets set synchronously with the store access, and further that the 'R' and 'M' bits get set atomically with respect to other system accesses to the page tables. Some implementations do not complete the update atomically, but rather implement an iterative algorithm to guarantee proper update of these bits.

**7.1.4 SRMMU Flushing**

Flushing is used to purge stale translations from the PDC. The flush operation is implemented through use of SPARC store alternate (*sta*) with ASI = 0x3. Flushes of user-mode pages (PTE[ACC] = 0-5) use the current contents of the Context Register (CTX). The address for a flush is composed as follows:



Field	Description												
VFA	<b>Virtual Flush Address:</b> Provides VA<31:12> for comparison on flushes												
TYPE	<b>Flush type</b> <b>Flush Object</b>												
	<table border="0"> <tr> <td>0 (page)</td> <td>Level-3 PTE</td> </tr> <tr> <td>1 (segment)</td> <td>Level-2 and -3 PTE/PTP's</td> </tr> <tr> <td>2 (region)</td> <td>Level-1, -2, and -3 PTE/PTP's</td> </tr> <tr> <td>3 (context)</td> <td>Level-0, -1, -2 and -3 PTE/PTP's</td> </tr> <tr> <td>4 (entire)</td> <td>All PTE/PTP's</td> </tr> <tr> <td>5 - 0xF</td> <td>None (reserved)</td> </tr> </table>	0 (page)	Level-3 PTE	1 (segment)	Level-2 and -3 PTE/PTP's	2 (region)	Level-1, -2, and -3 PTE/PTP's	3 (context)	Level-0, -1, -2 and -3 PTE/PTP's	4 (entire)	All PTE/PTP's	5 - 0xF	None (reserved)
0 (page)	Level-3 PTE												
1 (segment)	Level-2 and -3 PTE/PTP's												
2 (region)	Level-1, -2, and -3 PTE/PTP's												
3 (context)	Level-0, -1, -2 and -3 PTE/PTP's												
4 (entire)	All PTE/PTP's												
5 - 0xF	None (reserved)												
rsvd	<b>This field is ignored by the MMU, and must be zero.</b>												

Flush Type	PTE Flush Match Criteria (PTP criteria is same but no ACC checks)
page	level-3 and Addr[31:12] match and (PTE[ACC] = 6-7 or CTX equal )
segment	level-2 or -3 and Addr[31:18] match and (PTE[ACC] = 6-7 or CTX equal )
region	level-1 or -2 or -3 and Addr[31:24] match and (PTE[ACC] = 6-7 or CTX equal )
context	PTE[ACC] = 0-5 and CTX equal
entire	none (flush all PTE and PTP)

A 'precise' flush removes the minimum number of entries from the PDC. Implementations may choose to remove more entries than the minimum specified, that is, to implement an 'imprecise' flush.

Flushing is local to a module; that is, each IU is responsible for issuing flushes to its own PDC. During a demap operation, if more than one processor may have references to a stale PTE or PTP then each processor must issue flushes. Cooperative flushing may be implemented with directed *soft interrupts*.

### 7.1.5 SRMMU Probing

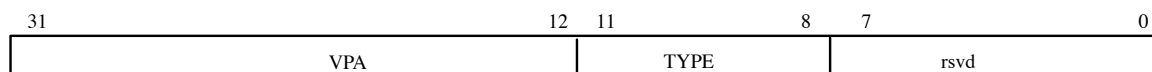
SRMMU probe operations return either an entry from a page table or the PDC, or generate an error. Probes are accomplished with a SPARC load alternate (*lda*) with ASI = 0x3. The current contents of the context register (CTX) are used. A probe operation will return the entry from a page table as implied by the TYPE field, or will return 0x0 if there is an invalid or translation error. A zero is likewise returned if a UE, TO, or BE is reported on a probe. These errors on a probe will not cause the processor to trap.

The requested entry is returned if it is valid (ET = 2) or invalid (ET = 0). The entry will also be returned if it is a PTP (ET = 1) except if it is found at level-3, in which case a zero is returned. A zero will be returned if an invalid (ET = 0) or reserved (ET = 3) entry is encountered at any intermediate level or if a UE, TO, or BE is reported on any access.

PROBE\_ENTIRE will return a valid PTE (ET = 2) from whichever level it is found at for the VPA and CTX, or will return a zero if no PTE is found or a bus error is experienced. There is no indication from a PROBE\_ENTIRE about which level the PTE was found at; determining the table level will require further probes of types 0 - 3, or software emulation of a table walk. PROBE\_ENTIRE is the only required probe type; all others are optional in an implementation.

Page, segment, and region probes should not update a PTE's Referenced bit, although implementations are allowed to set PTE[R] for PROBE\_ENTIRE.

The probe address is formed as follows:



Field	Description
<b>VPA</b>	<b>Virtual Probe Address: Provides VA&lt;31:12&gt; for table-walks on probes</b>
<b>TYPE</b>	<b>Probe type                      Probe Object</b>
	-----
	<b>0 (page)                      Level-3 PTE</b>
	<b>1 (segment)                  Level-2 and -3 PTE/PTP's</b>
	<b>2 (region)                  Level-1, -2, and -3 PTE/PTP's</b>
	<b>3 (context)                  Level-0, -1, -2 and -3 PTE/PTP's</b>
	<b>4 (entire)                    Level-n PTE's</b>
	<b>5 - 0xF                        None (reserved)</b>
<b>rsvd</b>	<b>This field is ignored by the MMU, and must be zero.</b>

### 7.1.6 SRMMU Diagnostic Access

Diagnostic read and write access to the entries in the PDC are done through the use of *lda* and *sta* accesses with ASI = 0x6, and with ASI = 0x5 if there are multiple SRMMU's in a single module. The details of what elements are accessed, how they are addressed, and what meaning is attached to specific bit fields is purely implementation dependent.

## 7.2 I/O Memory Management Unit (IOMMU)

### 7.2.1 IOMMU Overview

DVMA accesses will issue a virtual address on the SBus, which must be translated to a physical address in order to do the actual access. These translations are accomplished through a set of translation-look-aside-buffers (TLB's) in the MBus to SBus interface. The TLB's are part of the MMU used for DVMA. The IOMMU is a one-level MMU that is similar to a SPARC Reference MMU, but provides reduced functionality appropriate to DVMA needs.

The IOMMU is a one-level memory-based MMU; there is a base pointer that points to a table in memory, and part of the DVMA virtual address is used to index into this table to access an IOPTe (page table entry) which contains a physical page number, a 'valid' bit for the entry, a write-allowed bit, and a cacheability bit. The 'C' bit has no meaning for the DVMA interface, but is used on the MBus in support of DVMA coherence with the system data caches (see 7.3.2). Each IOPTe maps one 4KB page. The page table size and the corresponding DVMA virtual address range are configured in the IOMMU Control register RANGE field. The table consists of  $(DVMA\ Range / Pagesize)$  32-bit entries.

The IOMMU maps  $(DVMA\ Range)$  of virtual address space for DVMA activity. The virtual address used is  $VA\langle X:0\rangle$ , where 'X' is the highest VA bit in the translatable range.  $VA\langle 31:X+1\rangle$  must be all 1's in order for a translation to take place; otherwise an error is signalled to the DVMA master. The bits  $VA\langle X:12\rangle$  provide a virtual page number which is used as an index into the IOMMU table in memory, and the physical page number  $PA\langle 35:12\rangle$  is concatenated with the in-page index  $VA\langle 11:0\rangle$  to generate the physical address of the access. VME cannot access all of the virtual address range; see table 7.2.2.

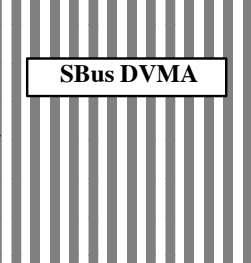

Intelligent DVMA masters may provide their own translation facility, and can bypass the IOMMU if the Bypass Enable bit is set in that device's slot configuration register. This facility is described in 7.2.5.1.

TLB entries in the IOMMU are allocated based upon an LRU algorithm. There are 16 TLB entries. TLB misses are serviced in hardware. Control/status registers are provided for management of the IOMMU. Translation faults are indicated with an error acknowledge to the DVMA master.

TLB flushes are done by writes to control space. Flushes involve writing a virtual address to the IOMMU flush register. If there is currently a TLB entry corresponding to that virtual page then the contents are discarded.

Diagnostic access is allowed when the IOMMU is disabled. The allocation logic and the data path can be exercised directly.

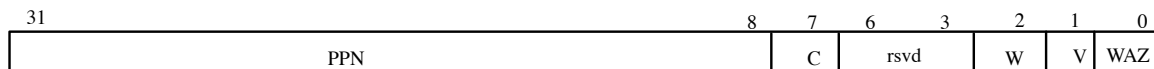
### 7.2.2 DVMA Virtual Address Space

DVMA VA	DVMA Address Source		
0x00000000			Invalid address range
DVMA base-1			
DVMA base -			DVMA Range
0xFF7FFFFFFF			
0xFF800000 - 0xFF8FFFFFFF			
0xFF900000 - 0xFFFFFFFF			

In this diagram, 'DVMA base' is the highest address in a 32-bit virtual space minus the DVMA range; DVMA addresses are always at the high end of the virtual address space.

The RANGE field in the IOMMU Control Register defines the virtual address range for DVMA. In order for a virtual address to be considered valid, all VA bits above the translatable address must be '1'. For example if RANGE = 2 then 64 MB of virtual addresses are supported, and valid DVMA virtual addresses range from 0xFC000000 to 0xFFFFFFFF. Any access using a DVMA virtual address that is out of the valid range will receive an error acknowledge. The only exception involves slots that have BYPASS MODE enabled; see 7.2.5.

### 7.2.3 IOMMU Page Table Entry (IOPTE) Format



Field	Description
PPN	<b>Physical Page Number: this field provides PA&lt;35:12&gt; for a translated address; this is concatenated with VA&lt;11:0&gt; to provide the entire translated address.</b>
C	<b>Cacheable: for coherent DVMA, this bit indicates if the IU entries to this data are cacheable. The bit is used in the MBus address and also triggers coherent access for accesses less than 32 bytes. If 'C' = 1 then the DVMA address must alias with the processor virtual address used to access this data.</b>
W	<b>Writeable: 1 = Read and write allowed, 0 = Read-only access.</b>
V	<b>Valid: 1 = Valid PTE</b>
WAZ	<b>Write as '0' always.</b>
rsvd	<b>Reserved.</b>

The IOMMU table consists of one 4-byte entry for each 4K page of address space that can be translated. The IBA field provides the base address of the IOMMU table. This table must be size-aligned; for example, if the RANGE = 2 then a 64 MB range of addresses is valid. Thus the table has 16K 4-byte entries for a total of 64 KB, and the table would have to be 64 KB-aligned. When forming the address of a PTE, the DVMA virtual address bits <X:12> are placed in PA<X-10:2>, PA<1:0> = 00, and PA<35:X-9> = IBA<35:X-9>, where X is the highest translatable bit of the DVMA virtual address. Again for example, if RANGE = 2 then the highest valid bit of the address is VA<25>, so the address of the PTE is PA<35:16> = IBA<35:16>, PA<15:2> = VA<25:12>, and PA<1:0> = 00.

This PTE is only accessed if the IOTLB does not already contain the translation. If the table is not properly aligned then unpredictable behaviour will result.

The IOMMU page table in memory must be treated as *non-cacheable* by the processor modules.

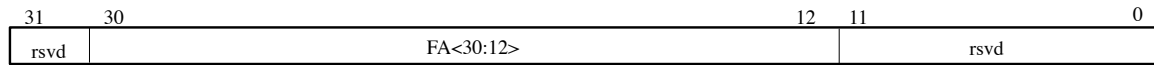
### 7.2.4 IOMMU Flushes

#### 7.2.4.1 Flush All TLB's: (PA = 0xFE000014)

Writes to the Flush All address will invalidate the current contents of all TLBs, independent of contents. This address is write-only.

#### 7.2.4.2 Address Based Flush: (PA = 0xFE000018)

Writes to the Address Based Flush register will cause any TLB with a tag corresponding to the IOPTE indicated by the FA to have its contents invalidated. This address is write-only.



Field	Description	Type
FA	<b>Flush Address: 32-bit virtual address of TLB flush</b>	<b>W</b>

### 7.2.5 SBus Slot Configuration Register

The SBus slot configuration register provides information about the slave device in that slot, and is also used for IOMMU bypass management for that slot. If an MBus processor attempts to access a slave and the size of that access is not supported, the M-to-S interface will break the access into a sequence of smaller accesses of a size that is supported. The actual sequence is implementation dependent; some implementations will automatically change non-supported bursts into a sequence of 32-bit words, and others may break the burst into smaller, supported bursts. If the access is broken into 32-bit words then normal SBus dynamic sizing is allowed to occur on each 32-bit access.

If an implementation supports wide-mode SBus devices, WMA enables both master and slave 64-bit activity for that slot. The boot code is expected to configure the slot based upon FCodes associated with the SBus device. (See 5.9 for a description of these registers)

#### 7.2.5.1 IOMMU Bypass Mode

Bypass mode is provided to allow intelligent SBus masters to do their own memory management with assistance from the kernel. It is assumed that such a master will have its own MMU. System performance will benefit from this if the intelligent master does not tend to local or sequential reference as normal DVMA does; a DVMA master that jumps around in large address spaces will impact DVMA TLB hit rates significantly. When bypass is enabled the DVMA master will issue bypass accesses as follows:

DVMA VA<31:30> = 1X: The VA is treated as a normal virtual address, and is translated by the IOMMU. This allows the master to use mapped accesses even with bypass mode enabled.

DVMA VA<31:30> = 00: The VA is treated as a bypass physical address. SEGA<35:30> is concatenated with VA<29:0> to form a 36-bit physical address, and the IOMMU is not used. The PA is checked to ensure that it is in the valid main memory range (PA<35:30> = 0x0), and an error is signalled to the master if it is not. In this mode, the access to main memory will be treated as non-cacheable (and not coherent) since the VA superset cannot be provided. In systems with physically-addressed processor caches, the CP bit is used to determine if the access should be treated as coherent in hardware. CP should always be set to '0' in systems with virtually-addressed processor caches. The SEGA field can be managed by the DVMA master if the valid main memory range exceeds 1 GB.

DVMA VA<31:30> = 01: Bits VA<29:24> are used to select a register or function within the interface that is accessible to the DVMA master. Currently VA <29:24> = 0 is supported, and any other value will cause an error acknowledge to the DVMA master. When VA<29:24> = 0, the DVMA master can access the slot configuration register associated with its own slot; the SEGA<35:30> and CP fields are RW, and the rest of the register is read-only for the DVMA master.

### 7.2.6 IOMMU Diagnostic Access

Diagnostic access to the IOMMU requires MMU off, DE = 1, and a quiescent DVMA system (no DVMA activity pending). Both the TAG array and the TLB array can be read and written in this mode; the CAM TAG array lives at 0xFE0000100 – 0xFE00013F; the contents are as shown in 7.2.6.2. The TLB RAM array lives at 0xFE0000200 – 0xFE000023F, and the contents defined in 7.2.6.1. In addition there are two diagnostic registers; the Diagnostic VA

Register at 0xFE0000140 (write-only) and the Diagnostic Comparator Output Register at 0xFE0000150 (read-only). When in diagnostic mode the value in the Diagnostic VA register bits <30:12> is compared to the contents of all tags; the comparator outputs can be viewed in bits <15:0> of the Comparator Output Register, with bit 0 corresponding to TLB #0, etc. A '1' indicates a match, and a '0' indicates a non-match. After diagnostic use, all TLB entries should have the 'V' bit set to '0' so that there are no inconsistencies under normal use. Care should be exercised if diagnostics are run with the the IOMMU enabled.

Writes to the Diagnostic Comparator Output Register are ignored. Reads of the Diagnostic VA register will return indeterminate data.

### 7.2.6.1 Translation Cache (TLB) Entry Format

31	8	7	6	3	2	1	0
PPN			C	rsvd	W	V	rsvd

Field	Description	Type
PPN	Physical Page Number	RW
C	Cacheable: sent with MBus address for coherence support	RW
W	Writeable: 1 = Read and write allowed, 0 = Read-only access.	RW
V	Valid: 1 = Valid TLB entry	RW
rsvd	Read as 0's, writing has no effect.	R

### 7.2.6.2 Translation Cache Tag Format

31	30	12	11	4	3	0
1	TAG<30:12>			rsvd	LRUQ	

Field	Description	Type
TAG	TAG <30:12> is compared to VA<30:12> on DVMA access. The high-order bits must be '1' (based on DVMA Range)	RW
LRUQ	TLB number in the Least-Recently-Used queue. The lowest address has the number of the LRU TLB, and the highest address has the number of the MRU TLB.	R
rsvd	Read as 0's, writing has no effect.	R

## 7.3 Cache Coherence

### 7.3.1 Level-2 MBus Support for Coherence

The level-2 MBus supports multiple-master coherence on a 32-byte block basis. At any time, a block in memory may have one or more copies in the system caches. If there are multiple copies the block is *'shared'*. If there is a single copy the data is *'exclusive'*. In the case of a block which a processor has written to, or is allocating with plans to write, the block is *'owned'*. A *shared* block may only be *owned* by one cache at any time. It is possible for a block marked *shared* to be the only copy currently cached.

MBus implements a write-invalidate policy. When a cache that has a *shared* block writes to that block, it broadcasts a Coherent Invalidate cycle on the MBus, which instructs all other caches that have a copy of that block to discard it, since a more recent copy exists elsewhere. If a cache is doing a write-allocate operation it issues a Coherent Read and Invalidate, which gets the most recent copy of that block and then causes all other caches to discard their copy. This makes the new block *exclusive* and *owned*.

The purpose of the *shared* status is to minimize the need for access to the MBus; *exclusive* blocks may be written to internal to the cache without waiting to send an *invalidate*.

If there are multiple virtual addresses that map to the same physical page, it is a rule that the OS must make the different virtual addresses map to the same cache line. This is the 'Virtual Address Aliasing Rule'. In virtual-address direct-mapped cache systems this means that the different virtual addresses must be identical modulo[1MB]. The 1MB rule is the absolute largest alias supported. Systems with smaller caches can use an alias rule based on the module cache size, which is specified in the module appendix. In physical address cache systems this rule does not apply.

Virtual address caches provide a 'flush' mechanism to support purging of references in the caches. This is implemented with flush ASI's as defined in 3.1.1.

MBus-based implementations of this architecture will use a cache block size or sub-block size of 32 bytes. This makes 32 bytes the *unit of coherence* in these systems.

### 7.3.2 The SPARC *flush* Instruction

It is important to note that the SPARC *flush* instruction affects instruction buffer consistency only on the processor that executes the *flush*. For cases where multiple modules may have buffered instances of an instruction that gets modified by a store executed by one processor, the execution of the *flush* will not guarantee that other processor modules have properly flushed their image of that instruction space.

This has no importance for common programs, but self-modifying code such as LISP or dynamic linkers may have an issue with this. It is a truism that multi-threaded applications with self-modifying code must have synchronization mechanisms around the threads to eliminate modify/execute race conditions; it is important that the synchronization mechanism includes local execution of *flush* in each thread for protection.

### 7.3.3 DVMA and Cache Coherence

By following certain rules in the allocation of DVMA addresses, consistency with the module caches can be maintained in hardware. This will reduce the need for cache flush cycles surrounding DVMA activity. In systems utilizing a level-1 MBus module, the module does not implement snooping, and software support via flushing is required to maintain coherence.

#### 7.3.3.1 Stream I/O Versus Non-Stream I/O

There are two classes of DVMA devices, stream and non-stream. Stream devices implement only sequential, unidirectional, block-sized and block-aligned transfers to memory. If such a device does a non-block-sized or non-block-aligned transfer then it is expected to access the entire block; that is, a write of less than 32 bytes will clobber the entire block. An example of a stream I/O device is the VMEbus I/O cache. Devices that implement stream I/O do not engage in fine-grained (cycle-by-cycle) coherence with the system caches; instead, the model is that pages are marked for transfer, the I/O transfer occurs, and then the pages are marked as available. During the time the transfer is taking place, those pages belong to I/O and the processors should not be allowed to access them. The kernel must enforce this rule.

Non-stream devices engage in fine-grained coherence. This means that upon each access to memory the hardware and kernel must ensure that both the processors and the DVMA device have access to the most recent copy of information at that address. In order to ensure this coherence, one of two models apply:

(1) If the page is marked non-cacheable in the SRMMU tables then it must also be marked non-cacheable (i.e. non-coherent) in the IOMMU tables. This method allows for any Virtual Page Number to be used by the DVMA access. The efficiency of the processor accesses to this page is poor.



(2) If the page is marked cacheable in the SRMMU tables then it must also be marked cacheable (coherent) in the IOMMU tables. In this case the DVMA virtual address must match the processor virtual address modulo[cache\_size]. *Hardware will ensure that coherence is maintained for both stream and non-stream devices under these models, although the processors must not touch pages belonging to a stream transfer during that transfer.*

The phrase 'must not touch' needs some explanation. If a processor writes to a page while DVMA is reading from it, or DVMA writes to a page while a processor is reading from it, then the data read may be indeterminate. For this reason, during stream model access these processor writes are strongly discouraged, and any processor read from a page during DVMA writes to that page are at risk. It is quite safe for both a processor and DVMA to be reading from the same page simultaneously.

The 1MB cache alias rule is due to the fact that the largest virtual coherent cache supported by the MBus is 1MB. If the size of the installed cache is less than 1 MB then the alias rule uses that cache size. Physically-addressed caches such as Viking do not require aliasing in order to provide DVMA coherence.

## 7.4 Write Buffers

Write buffers are used to accelerate writes and reduce bus occupancy for better overall system performance. While write buffers are not new to computer designs, some write buffers in Sun-4M implementations are by necessity visible to the software. Write buffers allow writes to complete concurrent with the processor doing useful work; the implication is that error reports are not synchronous to instruction flow. Errors are considered to be rare events, and should *never* be used for flow control.

All write buffers in the Sun-4M architecture follow these rules: (1) Once a write buffer has accepted a write, it must either guarantee that the write can occur without error, or the write buffer is responsible for reporting those errors. (2) Write buffers are read-stall; that is, after a write buffer has accepted a write, any subsequent access to that device must wait for the write operation to complete. (This ensures that order is maintained). That is, system write buffers are *strongly ordered*. (3) If a write buffer is visible to the software, it must have a synchronization mechanism; that is, software has a way of determining if a write is still pending or if it has completed.

Rule (2) does not apply to module write buffers, which are allowed to snoop cacheable items, but they must drain on *atomic* SPARC instructions or on read access to non-cacheable data. Module write buffers follow the TSO model defined in SPARC Architecture V.8; this is described in 7.5.

Write buffers exist in many places in implementations of this architecture, but only two are visible to software. Invisible write buffers are in the memory controller, the SCSI/Ethernet interfaces, and the E-bus interface; those write buffers will not ack until they have determined that the address and size are a valid transaction; then they will accept the write (if invalid they will error-ack; but the location of these buffers keeps that error-ack from reaching the modules. Instead, the previous write-buffer on the path will receive the error, and post an interrupt). Visible write buffers exist in the MBus-to-SBus interface and the VME Master port. There are also DVMA write buffers between SBus or VME/IOC and the MBus.

### 7.4.1 Write Buffers and Atomic Cycles

When an atomic load-store occurs either from a processor to the SBus or VMEbus, or a DVMA atomic cycle occurs to memory or to the SBus, that cycle will complete atomically. What this means is that the target bus will be held atomically between the read and write portions of the atomic cycle. The only thing that can break atomicity is if the target device issues a rerun on the write. Atomic cycles to devices that may issue an SBus *rerun* are not recommended.

### 7.4.2 Write Buffer Synchronization Support

The following synchronization mechanisms are provided for Sun-4M system write buffers: (1) memory interface: read EFSR or read from non-cacheable memory, (2) M-to-S: read the M-to-S AFSR, (3) VME Master: read VFSR, (4) DVMA to memory: do a DVMA read from memory. There are other write buffers that are not normally visible to software, but if there are time dependencies based upon completion of stores then the program should read back from the device that was written to. These write buffers exist in the counter/timer area (any read of any counter/timer register will suffice; that is, any register in the area PA<35:20> = 0xFF13), the interrupt logic block (any read of a register in the area PA<35:20> = 0xFF14), and on the path to other devices in the PA<35:24> = 0xFF1 space.

User code can always verify synchronization of writes to a device by reading back from that device; since devices are non-cacheable, the read-stall nature of write buffers will guarantee ordering.

### 7.4.3 DVMA Write Buffer Synchronization Support

When a DVMA write to memory has completed, or an IOC writeback or flush has been synchronized, that data may not be in memory yet due to a write buffer supplied for DVMA writes to memory. A DVMA master can synchronize these buffers by issuing a DVMA read of memory. *A processor module can synchronize the DVMA write buffer by reading the M-to-S AFSR three times.* It is expected that the kernel will have a routine called by all device drivers to synchronize the IOC and the DVMA write buffers after DVMA completes.

Write buffer synchronization does not work for the VME diagnostic loopback case; instead synchronization can be forced by issuing a VME loopback read that is not IOC-cacheable. Loopback is not used for normal operations.

## 7.5 Memory Model

Machines that are Sun-4M compliant will support the Total Store Order (TSO) model as defined in the SPARC Architecture Manual version 8. TSO guarantees that all store and load-store instructions from all processors are seen as being executed by memory serially; furthermore the stores and load-stores for a *particular* processor are made visible to the memory system in the order that they were issued by that processor.

A load by a processor may first check that processor's store queue to see if it contains a store to that address; if so that data is returned to the processor. If the data was not in the store queue, then the load operation goes directly to memory. Since not all loads go to memory, loads in general do *not* appear in the memory order. A processor may not issue another memory access until a load returns a value; until then the processor is *blocked*. (The check of the store queue is an implementation option, not a requirement under Sun-4M; the other choice is to stall until the store queue has drained.)

Atomic load-store (*swap* or *ldstub*) behaves like both a load and a store. The load-store is placed in the store queue like a store, and it blocks the processor like a load. Load-stores are thus treated atomically by memory. The processor blocks until the store queue is empty, so loads don't need to check for load-stores in the queue. Also, in TSO an atomic load-store is a point of strong synchronization, that is, when the load portion completes it is guaranteed that all previous stores have made it to the shared memory image.

It is recommended that shared-memory applications be written assuming the PSO (Partial Store Ordering) model defined in SPARC V.8 so that the same application will run across all SPARC MP platforms without modification.

Note that MMU table-walks will not participate in software locking schemes for access to MMU tables, which are shared memory entities.

## 8. The I/O Cache (IOC)

### 8.1 Overview of the IOC

The I/O cache in Sun-4M is provided to accelerate sequential VME slave port activity. Unlike previous IOC designs used at Sun, this IOC is not shared with SCSI or Ethernet traffic. IOC accesses to memory are always 32-byte bursts. On VME reads from main memory, leading fragments can be discarded. On VME writes to main memory that use the IOC, the writes will always start and end on 32-byte boundaries independent of the addresses used. Write-backs and flushes always trigger a 32-byte burst access. The IOC is not cycle-by-cycle coherent with main memory, but is coherent on a 32-byte burst basis. In SunOS this is known as the 'Streaming I/O' model.

The IOC is a write-back cache with a no-write-allocate policy. It is required that descriptors shared between the IU's and VME devices be non-IOC cacheable.

Each 32-byte line in the IOC will map to an 8KB section of VME address space; the mapping is a direct-map based upon VME A<22:13>. A total of 8MB of VME address space is allocated to the slave port, but only 1MB will respond in A24 space; the 1MB of A24 space overlays one of the 8MB of A32 space. Due to the 8K mapping, IOMMU entries for VME must be made identically on a pair of sequential 4K pages; that is, the write-allowed, cacheable, and valid bits must be the same, although the physical pages do not have to be physically contiguous.

VME slave port accesses can use or bypass the IOC on an 8K page basis.

The IOC does not participate in general SBus activity. SBus masters will achieve maximum bandwidth by issuing accesses that are system cache line-sized (32 bytes).

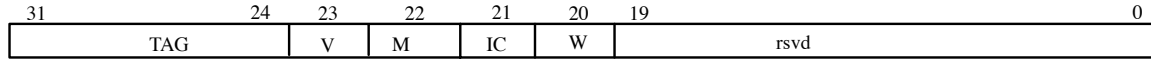
### 8.2 Management of IOC

The IOC requires software initialization, software flushes at the end of transfers, and tag initialization for each transfer. Cache initialization involves writing a '0' to the tag V and tag C bit in each tag. As each transfer from a VME device is established, the driver/kernel must establish valid mappings in the IOMMU entries corresponding to the VA allocated for the transfer, and must also write to the tag for each cache line allocated for the transfer. The tag write must (1) set V = 0, (2) set C = 1 if the transfer is IOC-cacheable, C = 0 if the IOC is bypassed for this 8K page, and (3) set the W bit to the same value as was set in the IOMMU. IOMMU entries must be mapped in pairs to correspond to the 8K IOC line mapping.

IOC flushes are required to complete transfers from VME to memory if the IOC is used. In normal operation the dirty line is written back to memory when another access to the line is attempted which misses on the tag; for this reason an IOC line will always contain valid data at the end of a write transfer, so the flush is always required. A flush will cause the corresponding IOC line to be written back to memory if the 'M' bit is set. This write-back is always a 32-byte burst, so flushes must not be used if the transfer through that cache line is still in progress.

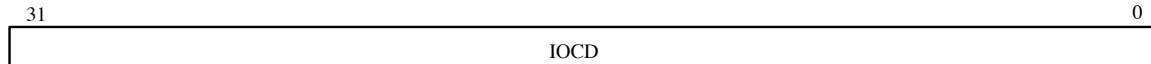
In order to trigger a flush of a cache line, the kernel must write to the IOC Flush address, with PA<14:5> selecting the cache line to be flushed. After a sequence of one or more flush writes, the kernel can determine that the flushes have completed by attempting to read from the tag memory. When the read is allowed to complete, the last flush has completed. This synchronization mechanism is managed in hardware.

**8.3 I/O Cache Tag Format (32-bit access)**



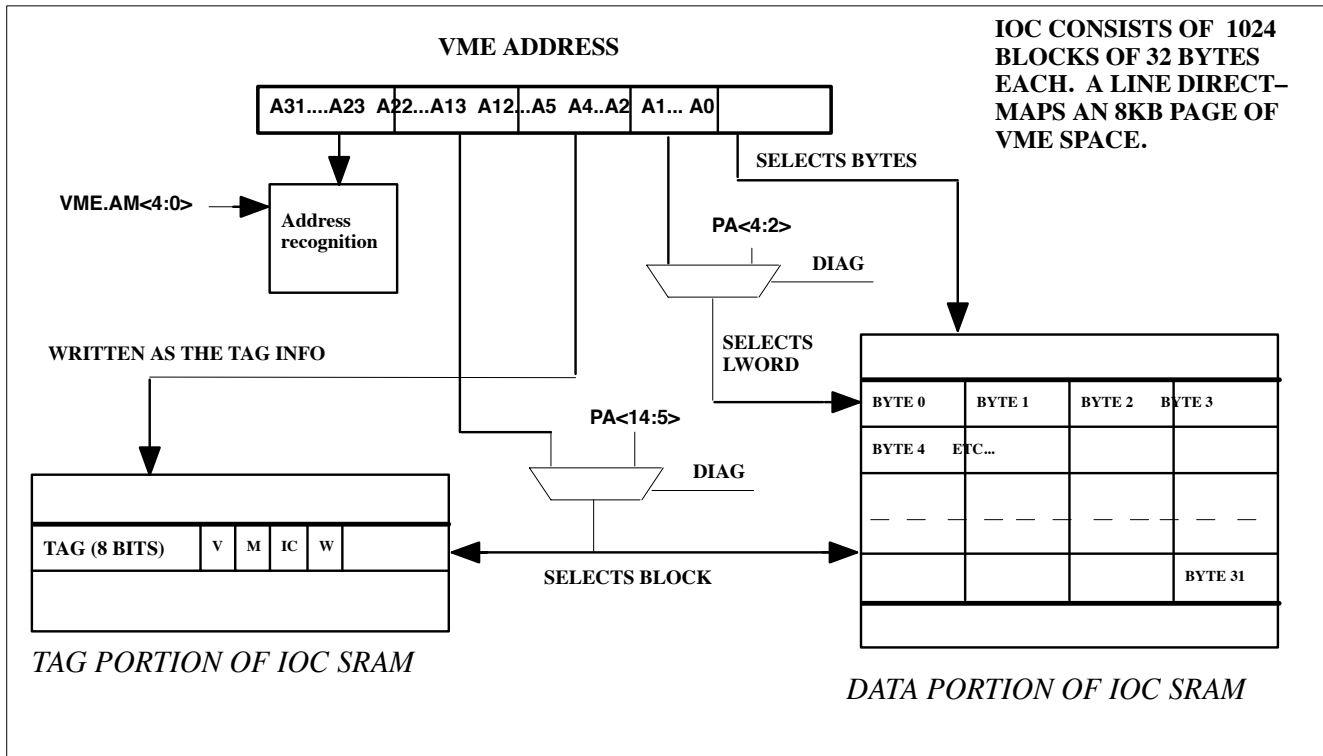
Field	Description	Type
TAG	Identifies block within 8K page; compared to VME.A<12:05>	RW
V	Valid bit: set when the tag contains valid information	RW
M	Modified: At least one byte of this cache line is dirty	RW
IC	IOC-cacheable: set by the kernel during mapping. This is independent of system cacheability of the data.	RW
W	Write-allowed: set by the kernel during mapping	RW
rsvd	reads as 0's, writing has no effect.	R

**8.4 I/O Cache Data Format, Diagnostic Access (32-bit access)**



Field	Description	Type
IOCD	Cache Data. PA<14:5> address the cache line, and PA<4:2> address the word within the 32-byte line.	RW

**8.5 I/O Cache Data Block Diagram**



## 9. I/O Devices

### 9.1 Keyboard/Mouse Interface and Serial Ports (8-bit access)

PA(35:00)	AMD Z85C30 Registers
0xFF100000 0xFF100002 0xFF100004 0xFF100006	Mouse Control Mouse Data Keyboard Control Keyboard Data
0xFF110000 0xFF110002 0xFF110004 0xFF110006	Serial Port B Control Serial Port B Data Serial Port A Control Serial Port A Data

Unspecified addresses in the PA<35:20> = 0xFF10 and 0xFF11 ranges are *reserved*.

The Keyboard/Mouse and Serial Port interfaces are each implemented using an AMD Z85C30 Serial Channel Controller. This device is software compatible with the Zilog 8530, but the recovery time is reduced from 1.6  $\mu$ S to 710 nS. An enhancement from previous Sun architectures is that the need for spin-loops is eliminated. As soon as a driver has finished an access to the serial port it is released to do useful work, and a subsequent access will stall if recovery time needs to be met. In a busy MP system this may impact system performance since the bus will be occupied by the stalled cycle; however in cases where high-speed serial port bandwidth is the primary goal, this will speed up the serial port activity.

Both devices will interrupt on level 12. See the AMD Z85C30 data sheet for further information.

### 9.2 EPROM (1, 2, 4, 8, 32-byte read access)

The EPROM consists of up to 1MB of PROM storage. The EPROM will mirror throughout the address space dedicated to EPROM (PA<35:24> = 0xFF0) independent of boot-mode or direct access. For example, if 512 KB of EPROM are supported then address bits PA<23:19> are ignored. If 128 KB of EPROM are supported then PA<23:17> are ignored.

Boot-mode accesses to the EPROM *must* have VA<27:24> = 0x0. EPROM is *not* cacheable.

### 9.3 TOD/NVRAM (8-bit access)

PA(35:00)	Mostek 48T08B
0xFF120000 – 0xFF1201Fd7 0xFF1201Fd8– 0xFF1201FF7 0xFF1201FF8 0xFF1201FF9 0xFF1201FFA 0xFF1201FFB 0xFF1201FFC 0xFF1201FFD 0xFF1201FFE 0xFF1201FFF 0xFF1202000– 0xFF12FFFFFF	NVRAM  "ID PROM" information (See 9.3.1)  Control register Second 00–59 Minute 00–59 Hour 00–23 Day 01–07 Date 01–31 Month 01–12 Year 00–99 Reserved.

**9.3.1 NVRAM/IDPROM (8-bit access)**

PA(35:00)	Mostek 48T08B
0xFF1201Fd8 0xFF1201Fd9 0xFF1201FdA – 0xFF1201FdF 0xFF1201FE0 – 0xFF1201FE3 0xFF1201FE4 – 0xFF1201FE6 0xFF1201FE7  0xFF1201FE8 – 0xFF1201FF7	<p><b>Format code:</b> indicates "IDPROM" format</p> <p><b>Machine Type</b> (see system 'A' appendix)</p> <p><b>Ethernet Address:</b> this unique 48-bit Ethernet address is assigned to the machine by Sun</p> <p><b>Date of manufacture:</b> format is a 32-bit word containing the number of seconds since January 1, 1970</p> <p><b>Serial number:</b> 3 bytes</p> <p><b>Checksum:</b> defined such that the longitudinal XOR of the 16 bytes from 0xFF1201Fd8 through 0xFF1201FE7 yields 0</p> <p>Assigned to manufacturing process use. <i>Format is TBD.</i></p>

This format is identical to the format of IDPROM's in the Sun-4 architecture, with the addition of an ECO-level field for manufacturing use. The use of NVRAM to contain IDPROM information was pioneered in the Sun-4c architecture.

**9.4 Audio/ISDN (8-bit access)**

PA(35:00)	AMD AM79C30A Registers	RW
0xFF1500000	<b>Interrupt Register (IR)</b>	R
	<b>Command Register (CR)</b>	W
0xFF1500001	<b>Data Register (DR)</b>	RW
0xFF1500002	<b>D-channel Status Register 1 (DSR1)</b>	R
0xFF1500003	<b>D-channel Error Register (DER)</b>	R
0xFF1500004	<b>D-channel Receive Buffer (DCRB)—8 byte FIFO</b>	R
	<b>D-channel Transmit Buffer (DCTB)—8 byte FIFO</b>	W
0xFF1500005	<b>Bb Channel Receive Buffer (BBRB)</b>	R
	<b>Bb Channel Transmit Buffer (BBTB)</b>	W
0xFF1500006	<b>Bc Channel Receive Buffer (BCRB)</b>	R
	<b>Bc Channel Transmit Buffer (BCTB)</b>	W
0xFF1500007	<b>D-channel Status Register 2 (DSR2)</b>	R

The audio interface in Sun-4M is identical to that in the Sun-4c architecture. The interface is provided through the Main Audio Processor (MAP) of the AMD 79C30A Digital Subscriber Controller. The 79C30A is a highly integrated circuit which provides an ISDN 4-wire subscriber level interface, an audio processing circuit, a parallel microprocessor interface, and a serial interface. For the audio interface, only the audio processing circuit and the microprocessor interface are used.

External to the 79C30A are an oscillator circuit using a 12.288 MHz, +/- 80 ppm, parallel resonant crystal and an operation amplifier used to drive the speaker.

The 79C30A interrupts on level-13. See the AM79C30A data sheet for further information.

**9.5 VMEbus Master Port (8, 16, 32-bit access. 32-bit not allowed in 16-bit spaces)**

The VME Master Port is accessed in one of 4 system address spaces. PA(31:00) is used as the VMEbus address. PA(35:32) provides control information.

<b>VMEbus Master Port Definitions</b>	
Address Sizes:	<b>A32, A24, A16</b>
	PA(31:00) = 0xFFFFxxxx is A16 space
	PA(31:00) = 0xFFxxxxxx is A24 space except for A16 space
Data Sizes:	<b>D32, D16, D8(EO)</b>
Interrupt Handler:	<b>IH(7-1), D8(O)</b>
Bus Timeout:	<b>&gt; 100 uS from master assertion of AS*</b>
Bus Requester:	<b>SGL, ROR</b>
Bus Arbiter:	<b>SGL, jumper disable</b>
Address Modifiers Generated:	<b>0x39, 0x3D, 0x09, 0x0D, 0x29, 0x2D</b>

The VME Master Port is accessed in one of 4 system address spaces. PA(31:00) is used as the VMEbus address. PA(35:32) provides control information. The VMEbus AM code always indicates a *data* access on the VMEbus signals.

PA(35:32)	VMEbus Space
<b>0xA</b>	<b>AM = User, 16-bit maximum transfer size</b>
<b>0xB</b>	<b>AM = User, 32-bit maximum transfer size</b>
<b>0xC</b>	<b>AM = Supervisor, 16-bit maximum transfer size</b>
<b>0xD</b>	<b>AM = Supervisor, 32-bit maximum transfer size</b>

PA(31:16)	VMEbus Space
<b>0xYYZZ</b>	<b>YY not = FF, AM = A32 space</b>
<b>0xFFZZ</b>	<b>ZZ not = FF, AM = A24 space</b>
<b>0xFFFF</b>	<b>AM = A16 space</b>

Atomic bus cycles are guaranteed atomic on the VMEbus; other activity may occur on the system busses during this time. In keeping with previous Sun VME implementations atomic cycles are implemented by holding VME BBSY\* between a load and a store, rather than by the VME atomic transaction protocol.

**9.6 SBus Expansion Slots**

Four SBus expansion slots are provided. Each is capable of being accessed as a slave. Slave capabilities are established in the SBus Slot Configuration Register for that slot. The SBus implementation must be compliant at least with SBus revision A.2; some implementations may be compliant with a more recent version of the SBus specification.

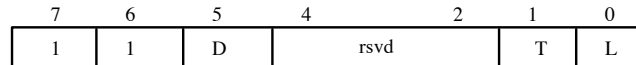
Depending on slave device capabilities, the SBus Slot Configuration Register (section 5.9) can be programmed to resize bursts into smaller transfers.

**9.7 Floppy Disk Controller (8-bit access)**

PA(35:00)	AMD AM82077 Registers	RW
0xFF170000	Status Register A ( <i>not used by Sun</i> )	–
0xFF170001	Status Register B ( <i>not used by Sun</i> )	–
0xFF170002	Digital Output Register (DOR)	RW
0xFF170003	Tape Drive Register ( <i>not used by Sun</i> )	–
0xFF170004	Main Status Register (MSR)	R
0xFF170005	Data Rate Select Register (DSR)	W
0xFF170006	FIFO Data Port (FIFO, 16-byte access)	RW
0xFF170007	reserved	–
0xFF170007	Digital Input Register (DIR)	R
0xFF170007	Configuration Control Register	W
0xFF170008 – 0xFF17FFFF	reserved	

The floppy will interrupt on level-11. See the AMD 82077 data sheet for further information. Note that the use of some register bits are different from those specified in that data sheet; this will be explained in the system appendices for systems that support floppy disk drives.

**9.8 Auxiliary I/O register 0 (8-bit access):** (*PA = 0xFF180000*)



Field	Description	Type
D	Floppy Density: 1 = high density, 0 = low density	R
T	Terminal Count: signals floppy drive that a transfer is done	W
L	LED (small systems): 1 = on, 0 = off.	W

All outputs reset to '0' on system reset. This register is provided on systems that support the Floppy Disk Controller. Bits that are 'reserved' may be specified in the system appendix for those systems.

**9.9 Generic I/O and Auxiliary I/O register 1 (8-bit access)**

There is support for some unspecified 8-bit slave device in some implementations. The support includes: at PA = 0xFF1A0000 to 0xFF1A00FF accesses can be made to an 8-bit slave device; and at address 0xFF1A0100 is an auxiliary I/O register; bits <7:4> are input, bits <3:0> are output. All outputs reset to '0' on system reset.

The use of these optional registers is defined in the system appendix for the systems that support them.

In the first implementation that is using them (Campus2) the AUXIO-1 register is used as a Power Control Register (see appendix A.II); the Generic Register is unused.

Implementations can choose to add more implementation-specific registers or devices on unique 4KB pages within the (PA<35:24> = 0xFF1) space.



## 10. DVMA and DMA Devices

### 10.1 SCSI and Ethernet Interfaces

Most Sun-4M implementations have on-board SCSI and Ethernet interfaces. Typically these interfaces are implemented as a single SBus device, and are configured to be the on-board SBus device at SBus slot 0xF (PA<35:28> = 0xEF). As an on-board device there is no SBus slot configuration register for this device. The SCSI/Ethernet implementation is described in the system-specific 'A' appendix.

### 10.2 VME Slave Port

VMEbus Slave Port Definitions
Address Sizes: <b>A32, A24</b> A32 responds to the lowest 8 MB of VME address space A24 responds to the lowest 1 MB of VME address space Data Sizes: <b>D32, D16, D8(E0); UAT and Bursts not supported</b> Interrupter: <b>No</b> Address Modifiers Recognized: <b>0x39, 0x3A, 0x3D, 0x3E, 0x09, 0x0A, 0x0D, 0x0E</b>

The VME slave port allows other VMEbus masters to access main memory through the DVMA mechanism. In A24 space the port will respond to VME addresses in the lowest MB of address space, and in A32 it will respond to the lowest 8 MB of space. (Note that prior Sun designs supported only 1 MB of space, so there may be addresses allocated to devices in A32, MB(7:1) that need to be moved). The VME address is used as a DVMA virtual address, with the address shifted to the highest 8MB of DVMA virtual address space (see 7.2.2).

The VME slave port supports an I/O cache for stream I/O; see section 7.3.2.1 and section 8 for further details. The IOC is direct-mapped, with one cache line allocated for each 8KB of VME address space. When a mapping is established for SVME access the IOMMU must have the correct PTE's for the address range to be accessed, the tag entries in the corresponding IOC lines must have the 'V' bit set to 0, and the 'C' bit must either be set for IOC use, or cleared for IOC bypass on that 8K range.

Data sizes of 8-, 16-, and 32-bit are supported for SVME transfers. Block-mode address modifiers are not recognized. VMEbus atomic cycles through the slave port are *not* guaranteed atomic on the system memory bus.

### 10.3 SBus Expansion Slots

'N' SBus expansion slots are provided. Each is capable of initiating a DVMA cycle, which can access main memory or another SBus slave device. The SBus implementation must be compliant at least with SBus revision A.2; some implementations may be compliant with a more recent version of the SBus specification. The limit on 'N' is based on SBus electrical characteristics and the number of on-board devices.

If an SBus master device contains its own memory management unit then its slot can optionally be configured to allow it to bypass the IOMMU. The configuration is accomplished with the SBus Slot Configuration Register (section 5.9) and is defined in section 7.2.5.1.

### 10.4 Valid DVMA Physical Addresses

DVMA devices are only allowed to access main memory (PA<35:32> = 0x0), MBus reserved spaces (PA<35:32> = 0x1-0x9) and SBus slaves (PA<35:32> = 0xE). Hardware will enforce this rule; if the IOMMU table contains a mapping with an invalid address then any DVMA access to that address will result in an error acknowledge to the DVMA master. See 7.2.2 for the VME and DVMA virtual address ranges.

## 11. Main Memory

### 11.1 Memory Overview

Main memory starts at physical address 0x0 and is not guaranteed to be contiguous. Memory size is probed according to the algorithm provided in the system-specific 'A' appendix. Memory addresses are not fully decoded; this means that within a row of memory, addresses will mirror if a smaller DRAM is used. Accesses to unpopulated rows of memory will return garbage data. The implementation notes for each Sun-4M machine will have guidelines for that machine's memory system.

Memory is ECC protected. ECC generation is always enabled; ECC checking can be disabled through the ECC Memory Control Register. Upon detecting a correctable error, the memory subsystem will correct the data delivered to the requester, and will also write the corrected data to memory. There is no hardware support for scrub. Diagnostic support for the ECC logic is provided.

### 11.2 Programming Notes on Software Scrub

Scrub of an ECC memory system is simply issuing a read access to each location over some period of time in order to catch soft errors. If an ECC error is detected it is required that the hardware will write the corrected data to memory. In a scrub cycle the data that is read from memory is discarded.

In a multiple cache environment care must be taken to ensure that a read done for scrub actually reads memory; in many cases a read will cause data to be issued from a cache that owns the block, rather than from memory.

A way to prevent this is to issue read cycles that are noncacheable when doing a scrub; this will both prevent a cache from responding, and also will prevent the data read from memory being snooped by caches in the system.

The following method is recommended for issuing SCRUB cycles. Set the Alternate Cacheable bit in the module control register to '0' (non-cacheable). In the generic specification section 4.1, this is bit 13 of the MCR; check the appropriate module specification to ensure that the module in use conforms with this. Then use the bypass ASI 0x20 to issue reads to the memory system; this ensures that the DRAM will respond, rather than a snooping cache. It is very important that the data read in this manner is not used or written to, since the multi-processor cache coherence could be destroyed.

The generic SCRUB code should issue sequential LDDA (addr) 0x20 to ensure that each location is read; the SCRUB code will access tables of valid physical addresses to determine loop boundaries. The SCRUB can be optimized in some systems; for example, if the memory system reads more than 64 bits on a non-cacheable read then the stride can be (size of read). Another optimization may take advantage of nonstandard block-read hardware to issue reads of 32-bytes, or use block-copy hardware to copy each location in memory to a single dump location. Each system will have different capabilities and performance impacts related to these.

Note that a correctable ECC error will cause a broadcast level-15 interrupt (if enabled in the ECC Memory Enable Register), so a flag should be posted when scrub is in progress so that the responding interrupt handler will know the source of the error.

## 12. Resets

There are several sources of reset in Sun-4M. These are Power-on Reset (POR), Software Reset (SWR), Reset Switch (RSTSW), and VME Reset In (jumper selectable; used only when the board is not the VME slot-1 master). In addition, each module may detect a Watchdog Reset if it experiences an error condition, which is a trap with traps disabled. This condition resets only the one processor, and generates a broadcast level-15 interrupt to the system. Each module may also receive a local software reset (SI) through that processor's Control Register; this facility is for diagnostic purposes only. Local software reset (SI) through the module control register will disable snooping for a long period of time, so it is not allowed during normal MP operations.

When a processor is reset, it needs to determine the source of the reset; the hierarchy it should search is local-watchdog, local SI, SWR, RSTSW, POR.

A reset is intended to put the processor or system into a known state. In order to allow for some robustness in system bringup, no state is reset in hardware unless it is absolutely necessary in order to ensure controllability. The contents of processor general registers, caches, tags, TLB's, and main memory are unaffected by RESET. All I/O devices and state machines will be reset; it is not possible to reset part of the system and leave the rest untouched.

The VME control register allows the kernel to issue a reset to the VMEbus in software, if the 'VME Reset Out' jumper is installed. A timing loop must be used to ensure that the VME reset is asserted for a minimum of 200 mS.

A reset switch is provided on-board for system bringup. This switch generates the equivalent of a power-on reset, with cache and memory contents preserved. The switch is not user accessible. A reset initiated with the switch will leave status in the System Control and Status Register.

Device, Bus, or Bit	State after POR or SWR or RSTSW	State after watchdog
Module: Caches Module MMUs 'Dual' bit (cache snooping) Boot-mode bit (1=boot mode, 0 = translate) SI bit Module write buffers Watchdog Bit	Disabled Disabled Off 1 0 Empty 0	unchanged unchanged unchanged 1 0 Drain normally 1
SWR_STAT RSTSW bit in System SCR SBus VMEbus Memory Controller Software Reset Status bit (SWR_STAT) Undirected_int_mask VME_Slave_enable Interrupt Target Register LED's IOMMU Soft-interrupt bits System Error bits System Write Buffers Counter/Timers	POR or RSTSW = 0, SWR = 1 POR or SWR = 0, RSTSW = 1 Reset Reset Reset '0' all '1' '0' 0x0 all ON Disabled all '0' all '0' Empty Initialized (see description)	unchanged

### **13. References**

- [1] SPARC Architecture Manual Version 8, 1/30/91, Sun Microsystems
- [2] SBus Specification Revision A.2, 1990: Sun Microsystems
- [3] MBus Specification, Rev 1.1, 1990: Sun Microsystems
- [4] The VMEbus Specification, Rev C.1, 10/85: Motorola
- [5] SPARC Multiprocessor Architecture, Faye Briggs and Michel Cekleov, Rev. 1.0 7/5/89
- [6] Sun Comet Architecture Committee Axioms (living document) (owner is Steve Chessin)
- [7] The Viking Microprocessor User Documentation, Rev 2.00, 11/01/90: Sun Microsystems
- [8] Viking Cache Controller (MXCC) Specification, (draft), 2/19/91: Sun Microsystems
- [9] SPARC RISC Users Guide, Second Edition, February 1990: Cypress Semiconductor
- [10] Suggested ASI Assignments for SPARC Systems, SPARC International, 9/13/88

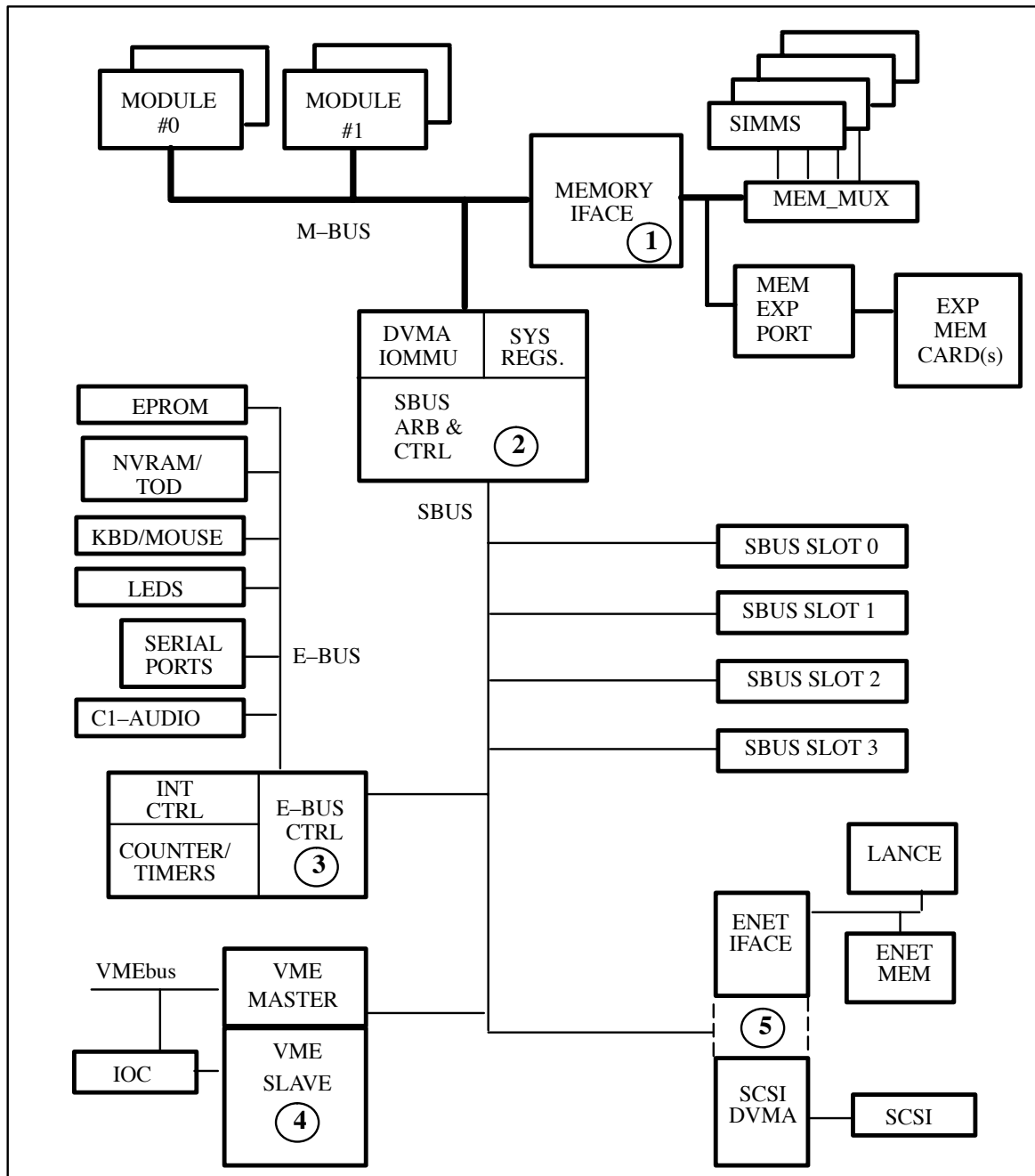
## Appendix A.I: Board Features: Galaxy

GALAXY is a single-board, high performance SUN workstation that supports up to four level-2 MBus processors on 2 MBus module connectors. Modules planned include Ross 6002 and Viking/E\$ on a Sun MBus. All processor modules installed must be of the same type.

- Double-height (2 slots) 9U form factor
- MBus running at 40 MHz, with two sockets. Each socket has support for a module containing one or two logical 'processor modules'.
- SBus interface at 20 MHz to low-cost graphics and peripheral devices
- 3 SBus expansion slots and one shared SBus/MBus slot (second processor module occupies the same space as the fourth SBus card)
- On-board memory of 16 or 32 SIMMS, ECC protected, interleaved, using 80 ns fast page-mode SIMMS of 1 or 4 MBit DRAM (support is included for future 16Mb parts also). Up to 128MB on-board with 4MB SIMMs.
- Expansion memory of up to 256 MB on each of two boards using 4 MBx9 SIMMs (system total of 640 MB with 4MB SIMMS, future 2.4GB with 16MB SIMMs)
- VMEbus master and slave ports with an I/O cache for the slave port
- LANCE Ethernet with local buffering
- Emulex ESP236 SCSI interface
- 'Sunness'; TOD/NVRAM, EPROM, keyboard/mouse, counter/timers, 12 diagnostic LEDs
- 2 standard 25-pin serial ports, synchronous-capable
- Campus-1 style audio interface
- Board Scan JTAG interface for manufacturing and field diagnosis

**A.I: System Specific Information: Galaxy**

**A.I.0: Galaxy Block Diagram**



(X) ASIC Partition: 1 = MMC, 2 = MSI, 3 = SEC, 4 = VIC, 5 = ESC

### A.I.1 SBus Details

Galaxy supports 4 SBus slots, numbered 0-3; each has the capability of being both a master and a slave. SBus dynamic sizing is supported for IU-initiated accesses. Slot configuration will allow for unsupported bursts to be converted to sequential 32-bit accesses only; those accesses can each dynamic size. The SBus supports up to 32-byte bursts for DVMA to/from memory and from the MBus. DVMA from SBus to SBus supports all burst sizes. There is also one on-board 'slot' at SBus slot 0xF, which contains a SCSI/Ethernet device.

The fourth slot shares space with the second MBus module. If the second MBus module is installed only three SBus slots are available.

SBus arbitration is fair per request; the requesters are slots<3:0>, on-board SCSI, and VME slave port. The SBus interconnect is also used to talk to the VME master port and the E-bus controller; this usage is not according to standard SBus protocols, but such usage is invisible to other devices on the bus.

DVMA coherence is supported on partial writes by the S-to-M interface, which allocates the line, merges in new data, and writes back to memory. This is not highly efficient; 32-byte burst devices are recommended for high bandwidth DVMA.

The M-to-S interface keeps track of reruns; if an SBus slave issues a rerun (which may be a stateful disconnect) the MSI tags that slave with the MID and a 'busy' bit; no other master is allowed to connect to the slave until the disconnected cycle is satisfied. If any SBus master is rerun from an SBus slave, that status is similarly recorded, but multiple SBus masters are not protected from each other if both are accessing the same slave device. Caveat Emptor.

DVMA from VME can access SBus slave devices as well as memory. SBus DVMA can access SBus slaves or memory also, but cannot access VMEbus slaves.

### A.I.2 MBus Details

Two MBus connectors are supported. Each has support for up to two MBus masters. Each connector can have a single-processor module, a dual-processor module, or one harvard-architecture module in it. The mappings to connector, MID, and IRL are as follows:

Connector	MID<3:0>	Non-Harvard	Interrupt	Harvard	Interrupt
0	1000	Processor 0	P0_IRL<3:0>	Processor 0 I-cache	P0_IRL<3:0>
	1001	Processor 1	P1_IRL<3:0>*	Processor 0 D-cache	-
1	1010	Processor 2	P2_IRL<3:0>	Processor 2 I-cache	P2_IRL<3:0>
	1011	Processor 3	P3_IRL<3:0>*	Processor 2 D-cache	-

\* this processor is only present if a dual-processor module is installed in the slot

MBus arbitration is fair among the modules, with DVMA at a fixed, higher priority than the module. The Galaxy MBus does not support modules that use 64-byte or 128-byte transfers. Wrapping is supported within read bursts.

The MBus *retry* acknowledge is never used, and MRDY\* acknowledges always provide good data; this allows for Viking, which uses data on the fly during cache fills. The memory controller will allow MIH\* from A+2 to A+8. Coherent Invalidate operations are acknowledged by the memory controller at A+3.

**A.I.3 Memory Details**

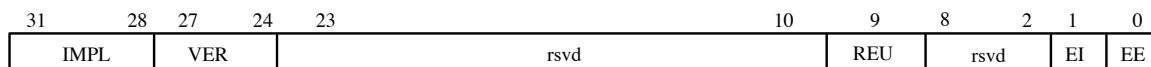
Galaxy has both on-board and off-board memory. The memory is ECC protected with SEC/S4ED codes. These codes provide the same coverage that is achieved with SEC/DED, plus detection of 3- and 4-bit errors within a nibble. Memory is accessed with 2-bank interleave; this makes the increment of upgrade 16 SIMMs, each 9 bits wide. 1MBx9, 4MBx9, and 16MBx9 80 nS fast page-mode commercial SIMMs are supported. All SIMMs in a row of 16 must be of the same type; thus each row can be 16 MB, 64 MB, or (far future) 256 MB. The expansion cards also has support for double-sided 8MBx9 SIMMs, and for 100 nS fast page-mode parts (for upgrades only).

The on-board configuration consists of two rows of 16 SIMMs each. The rows are based a 0x0 and 0x10000000 (256 MB boundaries). If SIMMs smaller than 16 MB are used, addresses will mirror within the 256 MB row. Off-board memory starts at 0x4000000 (1 GB) and is accessed and distributed in the same way.

Up to two memory expansion cards are supported. Each card has 64 SIMM sockets in four groups of 16. The first memory card contains rows of 256 MB of space starting at 0x40000000, 0x50000000, 0x60000000, and 0x70000000. The second card contains rows of 256 MB of space starting at 0x80000000, 0x90000000, 0xA0000000, and 0xB0000000. Either or both cards can be installed. The memory system requires use of a Sun Moonshine+ backplane interconnect; however the memory interface is not Moonshine+, but rather is special to Galaxy.

Each row of memory can be probed with write/write/read patterns at a 16-byte interval to determine if memory is installed in that row, and write/write/read on power-of-2 boundaries to see how much is there (by checking mirroring).

**A.I.3.1 ECC Memory Enable Register Difference from Sun-4M (32-bit access): (PA = 0xF0000000)**



Field	Description	Type
<b>EE</b>	<b>Enables ECC checking. Generation is always enabled. [1]</b>	<b>RW</b>
<b>EI</b>	<b>Enables Interrupt on correctable error. When '0' a CE will still be captured in the fault registers, but no interrupt will be generated. [1]</b>	<b>RW</b>
<b>REU</b>	<b>Memory Refresh Enable: a '1' enables refresh for the second row of on-board memory</b>	<b>RW</b>
<b>IMPL</b>	<b>Identifies the Sun Implementation of this memory controller = 0x0</b>	<b>R</b>
<b>VER</b>	<b>Version: Identifies the revision of this design = 0x0</b>	<b>R</b>

[1] clears to 0 on power-on reset.

If only one row is installed on the motherboard, performance can be improved by shutting off refresh to the (empty) second row. This is accomplished through bit<8> of the ECC Enable Register (5.5.2), which is a special function implemented on Galaxy. The bit is called REU (Refresh Enable Upper). When the bit is '1' all onboard memory receives refresh, when it is '0' only the first row receives refresh. This function is independent of off-board memory. This bit *must* be set to '1' if the second row is installed.

**A.I.3.2 SBus Slot Configuration Register Option: (Section 5.9)**

SEGA<35:32> are hardwired to 0x0.



### A.I.4 I/O Details and Options Supported

Galaxy supports an on-board LANCE Ethernet that accesses a local 128 KB buffer, and cannot DMA into system memory; this avoids Ethernet memory latency problems. SCSI is supported with the Emulex ESP236. There is no on-board video or floppy support. The diagnostic LED register supports 12 LED's in bits <11:0>; bits <15:12> are reserved. Galaxy supports 512 KB or 1MB of EPROM.

### A.I.5 Implementation ID numbers:

- (5.5.1) ECC IMPL = 0x0, ECC VER = 0x0
- (5.6.1) VME IMPL = 0x0
- (7.2.4) IOMMU IMPL = 0x0, VER = 0x1 (VER = 0x0 for obsolete prototypes)
- (9.2.1) Machine Type = 0x71
- SCSI/Ethernet TYPE = 0x4

### A.I.6 SCSI/Ethernet Interface Details

The Galaxy SCSI/Ethernet interface is implemented with the ESC SBus device. This chip provides interface to an Emulex ESP-236 SCSI controller and a LANCE Ethernet. The SCSI device is a DVMA master. The LANCE has a private 128KB buffer memory, and DMA's into that; the processor must transfer data to/from main memory and the buffer. This is done to prevent Ethernet dropped packets due to high memory latency.

#### A.I.6.1 Emulex ESP236 SCSI Port

##### A.I.6.1.1 SCSI Port Address Map

ADDRESS	REGISTER	TYPE	
0xEF0080000	Transfer Count Low	RW	<b>ESP registers (8-bit)</b>
0xEF0080004	Transfer Count High	RW	
0xEF0080008	FIFO Data	RW	
0xEF008000C	Command	RW	
0xEF0080010	Status/Bus ID	RW	
0xEF0080014	Interrupt Status/Timeout	RW	
0xEF0080018	Sequence Step/Synch transfer period	RW	
0xEF008001C	FIFO Flags/Synch. offset	RW	
0xEF0080020	Configuration	RW	
0xEF0080024	Clock Conversion Factor	W	
0xEF0080028	Reserved (test)	RW	
0xEF008002C	Configuration #2	RW	
0xEF0080030	Configuration #3	RW	
0xEF0080034- 0xEF0080FFF	Reserved	-	
0xEF0081000	SCSI DVMA Control Register	RW	
0xEF0081004	SCSI DVMA Address Register	RW	
0xEF0081008	SCSI DVMA Count Register	RW	
0xEF008100C - 0xEF0081FFF	Reserved	-	

##### A.I.6.1.2 SCSI DVMA Address Register

BIT	NAME	TYPE	MEANING
D<31:0>	A<31:0>	read-write	Virtual address used in SCSI DVMA access; bits <31:24> are register, bits <23:0> are counter.

**A.I.6.1.3 SCSI DMA Count Register**

BIT	NAME	TYPE	MEANING
D<31:24>	Reserved	R	Read as '0', writing has no effect
D<23:0>	COUNT	RW	SCSI DMA transfer count

**A.I.6.1.4 SCSI Control Register**

BIT	NAME	TYPE	MEANING	post RST
D<0>	SCSI_INT	R	ESP Interrupt is asserted	0
D<1>	ERR_INT	R	SCSI DVMA received an SBus ERR ack; clears on FLUSH	0
D<2>	BUF0	R	Buffer 0 has data (diagnostic)	0
D<3>	BUF1	R	Buffer 1 has data (diagnostic)	0
D<4>	EN_INT	RW	Enable INT_PEND to issue an interrupt to the system	0
D<5>	FLUSH	W**	Resets DVMA state machines. Do not assert if EN_DMA	0
D<6>	SLAVE_ERR	R	Slave access SIZE error, or access to ESP while RESET is asserted. Clears on read.	0
D<7>	RESET	RW	'1' = reset ESP and interface, '0' = normal ops.	1
D<8>	WRITE	RW	Direction of DVMA transfer *	0
D<9>	EN_DMA	RW	Allow DMA between ESP and interface	0
D<10>	REQ_PEND	R	Do not assert FLUSH or RESET while this is '1'	0
D<11>	BSIZE	RW	DVMA burst size: 0 = 32 byte, 1 = 16 byte	0
D<12>	TCZRO	R	Transfer count has expired; clears on load of counter.	0
D<13>	EN_TCI	RW	Enable interrupt upon terminal transfer count	0
D<14>	INT_PEND	R	Interrupt summary: SCSI_INT or ERR_INT or (TC0 and EN_TCI) or PERR	0
D<15>	PEN	RW	Enables SBus parity generation/checking	0
D<16>	PERR	R	Parity Error; st if PEN=1 and parity error; clear on FLUSH	0
D<17>	DRAIN	W**	Drains buffers to memory; poll BUF<1:0> for completion	0
D<18>	EN_AD	RW	Enables the Auto-Drain feature	0
D<27:19>	rsvd	R	Reserved, read as '0'	0
D<31:28>	ID	R	Interface Type = 0x4	0x4

\*MUST match direction of SCSI transfer. 1 = to memory, 0 = from memory

\*\*Writing a 0 has no effect.

Refer to the ESP-200 data sheet and the ESP-236 addendum for further details; also see the Galaxy Ethernet/SCSI Controller (ESC) ASIC spec, Sun P/N 950-1378-01, for a full programmer's model.

**A.I.6.2 LANCE Ethernet**

AMD ETHERNET INTERFACE			
PA<35:00>	REGISTER	SIZE	TYPE
0xEF0040000 – 0xEF005FFFF	LANCE buffer memory	1, 2, 4, 8, 32-byte	RW
0xEF0060000	Register Data Port	2-byte	RW
0xEF0060002	Register Address Port	2-byte	RW

The LANCE is able to DMA to/from the buffer memory only; it cannot DVMA to/from system memory. 128 KB of buffer memory are provided. Address bits above the 128 KB range are not decoded. Processor access to the buffer memory includes support for use of the processor *bcopy* hardware, which transfers blocks of 32 bytes.

Addresses in the range PA<35:28> = 0xEF that are not specified either here or in section A.I.6.2 are *reserved*. See the AM7990 LANCE data sheet for additional information.

### A.I.7 Bugs/Features

VME LOCK: If an IU issues a *ldstub* or *swap* to VME space, and the read portion receives either a BE or TO error (reported in the module SFSR/SFAR), the VMEbus will remain locked. Unlocking it requires a read of the VFSR, discarding the data read. Since it is not clear that this (atomic) event occurred, the VFSR should be read after any read with BE or TO that is mapped to VME space.

MID Register: A bug in the first implementation of the MID register makes this function unusable in this system. The description of the MID Register in section 5.4.3 describes general workarounds for this bug.

### A.I.8 Board Partition

There are 6 ASICs in the Galaxy design. The MSI (MBus/SBus Interface) implements the M-to-S and S-to-M functions, including the IOMMU, write buffers in both directions, the SBus arbiter, and the MBus arbiter. The M-to-S asynchronous error registers and the arbiter enable register reside in this chip, along with all SBus-related and IOMMU related registers. The MSI is also intended for use in Campus2.

The MMC (Main Memory Controller) interfaces the MBus to main memory both on- and off-board. It contains ping-pong write buffers, ECC generation and check logic, and the ECC error registers. It also generates the controls to the memory system. The MUX ASIC provides 2-bank interleave access and address buffering for the memory system in 9-bit slices; it is used both on-board and on the memory expansion board.

The SEC (SBus/EBus Controller) implements the Sun-4M multiprocessor interrupt logic, and interfaces the system to the 8-bit slave I/O on the E-bus (*'E' stands for 'Eight-bit'. Well, what did you expect?'*). The SEC is accessed through the SBus, although it is not an SBus device. The SEC uses a special multiplexed mode to receive address information from the MSI on the S\_D bus, and does not connect to the S\_A bus. It also contains the system status and control register. The SEC is also intended for use in Campus2; some pins that are not used on Campus2 (VME interrupts, P2\_IRL and P3\_IRL) are remapped via the C2\_MODE pin into interface to a floppy controller, a video controller, and a generic 8-bit slave device.

The ESC (Ethernet and SCSI Controller) is an SBus device that interfaces the SBus to an on-board LANCE Ethernet with local buffer memory, and to an Emulex ESP236 SCSI interface. This same ASIC can be used on an SBus expansion card. It supports parity on the SCSI/SBus function for use on Sundragon.

The VIC (VMEbus and I/O cache Controller) provides both master and slave interface to a VMEbus. The slave port supports up to 8MB of VME address space, and has an I/O cache for accelerating sequential activity. As a VME slave the VIC behaves as a normal SBus DVMA master. The VME master port is accessed over the SBus using the same special multiplexing as the SEC uses. It also receives some additional non-SBus information such as LOCK.

### A.I.9 Official Product Designations

The CPU board by itself is called the Sun 4600MP CPU Board. The products based on this board are known collectively as the SPARCsystem 600MP Series. The word 'system' is replaced with 'station' or 'server' depending upon configuration. Different package designators make the specific systems the 630MP, the 670MP, and the 690MP.

## A.II: System Specific Information: Campus-2

*This appendix is based on the Campus-2 Programmer's model version 1.02, the Campus-2 Theory of Operations (DRAFT) version 0.4, The Campus-II DMA2 Chip Specification version 6.0, the Campus-II Video Interface System Specification version 1.0 (6/25/90), and innumerable conversations with members of the Campus-2 design and operating system groups.*

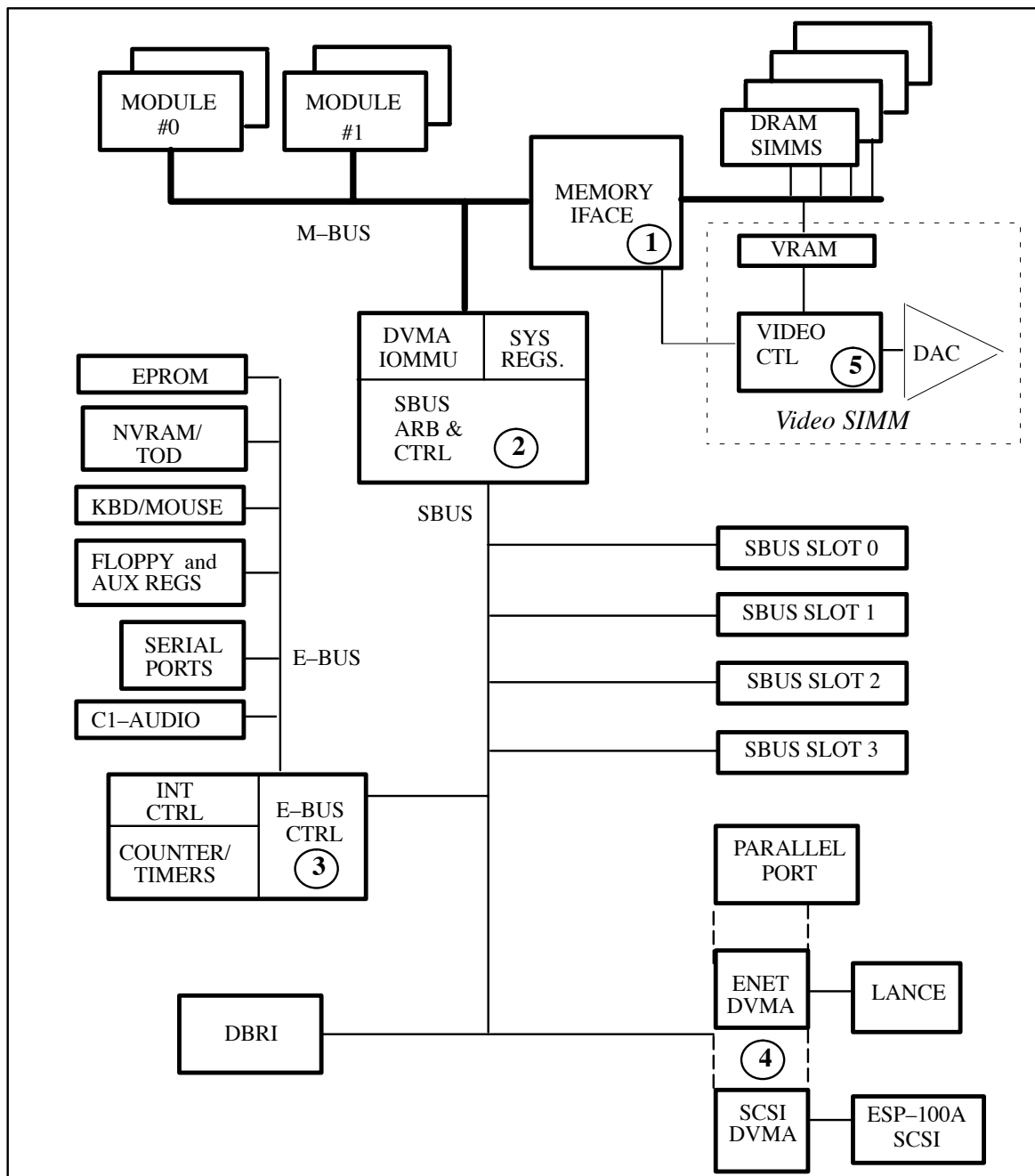
*Details in this appendix are subject to change. For up-to-date information contact the Campus-2 product team. People with **Need-to-Know** can obtain copies of the relevant specifications from Janice Bater (janiceb@Eng), subject to approval by project management.*

Campus-2 is a single-board, high performance SUN workstation that supports up to four level-2 MBus processors on 2 MBus module connectors. Modules planned include Ross 6002, Viking/NE, and Viking/E\$ on a Sun MBus. All processor modules installed must be of the same type.

- 'Pizza box' form factor
- MBus running at 40 MHz, with two sockets. Each socket has support for a module containing one or two logical 'processor modules'.
- SBus interface at 20 MHz to low-cost graphics and peripheral devices
- 4 SBus expansion slots
- On-board memory of 1-8 SIMMS, ECC protected, interleaved, using 80 ns fast page-mode SIMMS of 1 or 4 MBit DRAM (support is included for future 16Mb parts also). Up to 512MB on-board with 64MB SIMMS. These SIMM's are custom to the Campus-2 project.
- Optional Video SIMMs (VSIMM's) supported for low-cost frame buffer in memory space
- LANCE Ethernet
- Emulex ESP100A SCSI interface
- On-board Dual Basic Rate ISDN (DBRI) interface
- 'Sunness'; TOD/NVRAM, EPROM, keyboard/mouse, counter/timers.
- 2 standard 25-pin serial ports, synchronous-capable, in a single DB25 connector
- Campus-1 style audio interface
- Board Scan JTAG interface for manufacturing and field diagnosis for some ASICs
- Type-5 Sun Keyboard
- *A future version of this machine (Campus-2+) will incorporate integrated SPAM graphics. That version will be described in another appendix when the details are firm.*

**A.II: System Specific Information: Campus-2**

**A.II.0: Campus-2 Block Diagram**



(X) ASIC Partition: 1 = EMC, 2 = MSI, 3 = SEC, 4 = DMA2, 5 = MDI

### A.II.1 SBus Details

Campus-2 supports 4 SBus slots, numbered 0-3; each has the capability of being both a master and a slave. SBus dynamic sizing is supported for IU-initiated accesses. Slot configuration will allow for unsupported bursts to be converted to sequential 32-bit accesses only; those accesses can each dynamic size. The SBus supports up to 32-byte bursts for DVMA to/from memory and from the MBus. DVMA from SBus to SBus supports all burst sizes.

There are also two devices attached to one on-board 'slot' at SBus slot 0xF, which contains a SCSI/Ethernet/Parallel port device and a Dual Basic Rate ISDN (DBRI) device respectively. Note that the Arbiter Enable register bit <20> enables arbitration for the SCSI/Ethernet/Parallel Port device, while there is no way to disable arbitration for the DBRI device.

SBus arbitration is fair per slot request; the requesters are slots<3:0>, on-board SCSI/Ethernet/Parallel Port, and the DBRI Codec. The SBus interconnect is also used to talk to the E-bus controller; this usage is not according to standard SBus protocols, but such usage is invisible to other devices on the bus.

DVMA coherence is supported on partial writes by the S-to-M interface, which allocates the line, merges in new data, and writes back to memory. This is not highly efficient; 32-byte burst devices are recommended for high bandwidth DVMA.

The M-to-S interface keeps track of reruns; if an SBus slave issues a rerun (which may be a stateful disconnect) the MSI tags that slave with the MID and a 'busy' bit; no other master is allowed to connect to the slave until the disconnected cycle is satisfied. If any SBus master is rerun from an SBus slave, that status is similarly recorded, but multiple SBus masters are not protected from each other if both are accessing the same slave device. Caveat Emptor.

SBus DVMA can access SBus slaves or main memory.

### A.II.2 MBus Details

Two MBus connectors are supported. Each has support for up to two MBus masters. Each connector can have a single-processor module, a dual-processor module, or one harvard-architecture module in it. The mappings to connector, MID, and IRL are as follows:

Connector	MID<3:0>	Non-Harvard	Interrupt	Harvard	Interrupt
0	1000	Processor 0	P0_IRL<3:0>	Processor 0 I-cache	P0_IRL<3:0>
	1001	Processor 1	P1_IRL<3:0>*	Processor 0 D-cache	-
1	1010	Processor 2	P2_IRL<3:0>	Processor 2 I-cache	P2_IRL<3:0>
	1011	Processor 3	P3_IRL<3:0>*	Processor 2 D-cache	-

\* this processor is only present if a dual-processor module is installed in the slot

MBus arbitration is fair among the modules, with DVMA at a fixed, higher priority than the module. The Campus-2 MBus does support modules that use 64-byte or 128-byte transfers to memory but not to the SBus. *Sun currently has no plans for modules that issue accesses larger than 32 bytes.* Wrapping is supported within read bursts.

The MBus *retry* acknowledge is never used, and MRDY\* acknowledges always provide good data; this allows for Viking, which uses data on the fly during cache fills. The window for MIH\* is programmable up to A+14, but anything that requires MIH\* later than A+8 will slow down general memory performance. The memory controller will programmably acknowledge a CI at A+2 to A+9, or will ignore them if a coherent interconnect is plugged in and has taken ownership of CI acknowledge.

### A.II.3 Memory Details

Campus-2 supports on-board memory only. The memory is ECC protected with 8-byte SEC/S4ED codes. These codes provide the same coverage that is achieved with SEC/DED, plus detection of 3- and 4-bit errors within a nibble. Memory is implemented with custom SIMM's that provide a 144-bit data path (16 bytes plus 16 check bits). The increment of upgrade is one SIMM. SIMM sizes are 16MB (1 M x 16 byte), and (future, possibly at FCS) 64 MB (4M x 16 bytes). SIMMS of different types can be mixed in a system. Some SIMM sockets may contain a VRAM SIMM frame buffer instead of DRAM.

The memory configuration consists of 8 SIMMs. The sockets are at 64MB boundaries, so memory must be probed on 64MB boundaries. If SIMMs smaller than 64 MB are used, addresses will mirror within the 64MB slot space.

If any SIMM sockets are not populated, performance can be improved by shutting off refresh to the empty rows. This is accomplished through bits<9:2> of the ECC Enable Register (section 5.5.2), which is a special function implemented on Campus-2. The bits are called MRR<7:0> (Memory Row Refresh). When the bits are '1' the corresponding socket receives refresh, when they are '0' it does not receive refresh.

The following algorithm is provided by the Campus-2 team for identifying VRAM and DRAM configurations: The following transfer sequence is an example of identifying the SIMM in SLOT #7 after system reset. This algorithm can be repeated for each inserted SIMM by adjusting the slot specific VCONFIG value and the memory transfer addresses a[28:26] = logical slot number:

```

Store MER a=0xF0000000 d=0x000003FC      /* Disable ECC, Enable refresh */
Store VCONFIG a=0xF0000000C d=0xC000
Store a=0x0FC000000 d=0x1111
Store a=0x0FC400000 d=0x2222
Store a=0x0FC800000 d=0x3333
IF (({Load a=0x0FC400000} == d=0x2222) & ({Load a=0x0FC800000} == d=0x3333))
    IF ({Load a=0x0FC000000} == d=0x1111): SLOT #7= 16MB HRTC VSIMM
    ELSE: SLOT #7= 8MB HRTC VSIMM
ELSE
    Store VCONFIG a=0xF0000000C d=0x8000
    Store a=0x0FC001000} == d=0x4444
    IF ({Load a=0x0FC000000} == d=0x3333): SLOT #7= 4MB VSIMM
    ELSE
        Store VCONFIG a=0xF0000000C d=0x4000
        IF ({Load a=0x0FC000000} == d=0x4444): SLOT #7= 2MB VSIMM
        ELSE
            Store VCONFIG a=0xF0000000C d=0x0000
            Store a=0x01C000000 d=0x5555
            Store a=0x01C800000 d=0x6666
            Store a=0x01E000000 d=0x7777
            IF ({Load a=0x01C000000} == d=0x5555): SLOT #7= 64MB DSIMM
            ELSEIF ({Load a=0x01C800000} == d=0x6666): SLOT #7= 16MB DSIMM
            ELSEIF ({Load a=0x01E000000} == d=0x7777): SLOT #7= 4MB DSIMM
            ELSE: SLOT #7= >EMPTY<

```

*Note that physical row number does not correspond to logical row number.* Physical sockets 0-1-2-3-4-5-6-7 correspond to logical address spaces 0-4-1-5-2-6-3-7. Physical slots 1 and 3 can contain either a DRAM SIMM or a VRAM SIMM. The VRAM SIMM has no ECC, and provides a frame buffer in main memory space. Slots that have a VSIMM installed should have refresh disabled in the MRR field. Physical slots 5 and 7 also support VSIMMs but have no video extension socket.

**A.II.4 Implementation-Specific Registers and Bits in Sun-4M Registers****A.II.4.1 Additional Fields in Sun-4M Registers****A.II.4.1.1 ECC Memory Enable Register (32-bit access): (PA = 0xF0000000)**

31	28	27	24	23	12	11	10	9	2	1	0
IMPL	VER	rsvd				DCI	A	MRR<7:0>	EI	EE	

Field	Description	Type
EE	Enables ECC checking. Generation is always enabled. [1]	RW
EI	Enables Interrupt on correctable error. When '0' a CE will still be captured in the fault registers, but no interrupt will be generated. [1]	RW
MRR<7:0>	Memory Refresh Enable: a '1' enables refresh for the corresponding row of DRAM (logical socket number)	RW
A	Reads as '0' for this implementation	R
DCI	Disables Coherent Invalidate ACK; set to '0' for normal operation	RW
IMPL	Identifies the Sun Implementation of this memory controller = 0x1	R
VER	Version: Identifies the revision of this design = 0x0	R
rsvd	Reads as 0's, writing has no effect.	R

[1] clears to 0 on power-on reset.

**A.II.4.1.2 ECC Fault Status Register Differences (PA = 0xF0000008)**

Bit <1>: SE, Slot Error. VRAM & VIO access to a DSIMM, or DRAM access to a VSIMM.

Bit <2>: TO, Offboard Timeout on write: *Not supported, no offboard memory.*

Bit <17>: GE, Graphics Error. UE or CE during a graphics access. *Not used in this version of Campus-2.*

**A.II.4.1.3 ECC Diagnostic Register (PA = 0xF0000018)**

Bits<11:10>: DMODE. Campus-2 does not support the choice of forced diagnostic store or diagnostic load check bits; instead, the field has bit <11> as 'reserved', and bit <10> is defined as 0 = normal mode, 1 = diagnostic mode. In diagnostic mode, a store will cause CB<7:0> to be stored in memory in place of the generated check bits; a load will cause the CB<7:0> field to be loaded in the syndrome field SYN<7:0> of the EFSR.

**A.II.4.1.4 Diagnostic Message-Passing Registers (PA = 0xF00001000 - 0xF00001003)**

Campus-2 does not provide these registers.

**A.II.4.1.5 AUXIO 0 Register (PA = 0xFF1800000)**

Campus-2 defines the following additional bits:

Bit <4>: E: Edgeon, ro. When '1', indicates that a manufacturing test jumper block is installed; should be removed for shipped products.

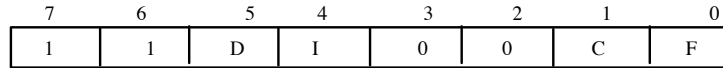
Bit <3>: S: SCCB-IMUX, wo. When '0' serial port B receives data from the REC\_DATA\_B pin on the serial port connector; when '1' serial port B receives the differential MIDI data from the Audio/AUI connector.

Bit <0>: L: LED\_ON, wo. Campus-2 uses this single LED instead of the LED register.

**Important note about this register:** *AUXIO-0 is shared between the floppy, LED, and serial-port drivers. Since there are shared functions and write-only bits, this register must be shadowed in software and the shadow copy must be made both interrupt- and MP-safe.*



**A.II.4.1.6 AUXIO 1 Register (Power Control Register):** (PA = 0xFF1A0100)



Field	Description	Type
<b>D</b>	<b>Power Fail detect.</b> When '1', a power failure has been detected. Assertion of this bit causes a <code>module_error</code> broadcast interrupt to be sent to all processors. Resets to '0'.	<b>R</b>
<b>I</b>	<b>Keyboard Power-on Interrupt.</b> When '1' indicates that the keyboard device has sent a power-on request, and a SBus-L7 interrupt is asserted (SPARC IRL13).	<b>R</b>
<b>C</b>	<b>Inhibit Kbd Power-on Interrupt.</b> When '1', enables bit I to assert and to cause an interrupt. Writes to this register with this bit '0' will clear/hold bit I at '0'.	<b>S</b>
<b>F</b>	<b>Power Off.</b> Writes with this bit '1' turns off power to the system	<b>W</b>

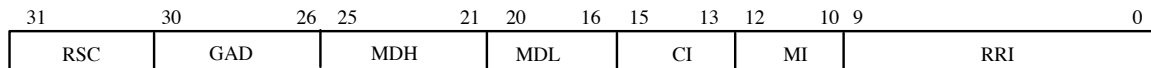
Bit 'I' gets set if the type-5 keyboard power-on button is pressed while the system is powered up, if bit 'C' = '1'. The action to be taken in this case is TBD, probably to issue a soft power-down after verification.

**A.II.4.1.6 SBus Slot Configuration Register Option:** (Section 5.9)

SEGA<35:32> are hardwired to 0x0.

**A.II.4.2 Additional Implementation-Specific Registers in this Memory Control Space**

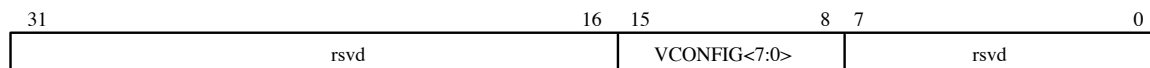
**A.II.4.2.1 Memory Delay Register (32-bit RW Access):** (PA = 0xF0000004)



Field	Description	Type
<b>RSC</b>	<b>Refresh load control; normally set to '1', exists for diagnostics.</b>	<b>W</b>
<b>GAD&lt;4:0&gt;</b>	<b>Graphics Arbitration Delay: when '0' buffered MBus transactions always win. †</b>	<b>RW</b>
<b>MDH&lt;4:0&gt;</b>	<b>MBus master (0xF-8) arbitration delay from graphics †</b>	<b>RW</b>
<b>MDL&lt;4:0&gt;</b>	<b>MBus master (7-0) arbitration delay from graphics †</b>	<b>RW</b>
<b>CI&lt;2:0&gt;</b>	<b>Coherent Invalidate Delay: number of clocks after A+2 that the memory controller will ACK a Coherent Invalidate cycle.</b>	<b>RW</b>
<b>MI&lt;2:0&gt;</b>	<b>MIH Delay: Cycle after A+7 that first MRDY* is issued.</b>	<b>RW</b>
<b>RRI</b>	<b>Refresh Request Interval: number of clocks between refresh requests to different rows. Default value is 0x20.</b>	<b>RW</b>

The default value to put into this register for Campus-2 with either the Ross605/64K (B.II) or the Viking/NE (B.I) is 0x00002095. With the Viking/E\$ module (B.III) the value should be 0x00002895.

† GAD<4:0>, MDH<2:0>, and MDL<2:0> are not used in this version of Campus-2.

**A.II.4.2.2 Video Configuration Register (32-bit RW Access):** (*PA = 0xF000000C*)

Field	Description	Type
VCONFIG<1:0>	VIDCONF<4><1:0>: Control bits for the SIMM in logical slot 4	RW
VCONFIG<3:2>	VIDCONF<5><1:0>: Control bits for the SIMM in logical slot 5	RW
VCONFIG<5:4>	Control bits for the SIMM in logical slot 6 ( <i>Not supported in C2</i> )	RW
VCONFIG<7:6>	Control bits for the SIMM in logical slot 7 ( <i>Not supported in C2</i> )	RW

VIDCONF<x><1:0>	Description	Page
00	DRAM or Empty slot	–
01	128-bit 2MB frame buffer	4K
10	128-bit 4MB frame buffer	8K
11	256-bit 8/16MB frame buffer	16K

The VCR is used to configure the EMC to properly manipulate the address of frame buffer accesses. It is used in the probe algorithm as described in section A.II.3, and should be set to the correct value for all slots when configuration is complete. Since logical slots 6 and 7 do not support VSIMMs in this machine the bits VCONFIG<7:4> should always be '00'.

*SIMM sockets 6 and 7 support SIMMs that implement the VSIMM protocol but do not have support for connection to an external video connector. Currently there are no plans for special SIMMs that take advantage of this feature.*

**A.II.4.2.3 Video Controller (MDI) Register Addressing**

If a VSIMM is installed then the address space for that slot is mapped to the control and access of a frame buffer implemented with an MDI (Memory Display Interface). The MDI is a video display controller with up to three 24-bit color look-up tables (CLUT's), hardware cursor support, transparent overlay with bleeding, and fully programmatic monitor timing. The MDI will process 8-bit greyscale, 8-bit pseudo-color, 16-bit (8+8) and 32-bit pixels.

MDI is designed to support 76 Hz non-interlaced video. A 4 MB VSIMM will support 1152x900 pixels with 32-bit true color; the same resolution will be supported at 84 Hz. An 8 MB VSIMM will support either 1280x1024 or 1600x1280 pixels at 76 Hz.

Details of the MDI functionality, registers, and addressing can be found in *Campus-II Video Interface Specification version 1.0*. The UART function described in that specification is not implemented. The initial implementation of the MDI has the Revision Status Register = 0x00: Major Revision 0x0, Implementation = 0x0 (2 CLUT's, Double CLUT RAM mapping). See that specification for further information.

VSIMM	Base Address	Control Space Base Address
SIMM 4	0x01000000	0x09000000
SIMM 5	0x01400000	0x09400000
SIMM 6 †	0x01800000	0x09800000
SIMM 7 †	0x01C00000	0x09C00000

† Not a video SIMM, but this slot support non-DRAM special SIMMS (*future*)

**A.II.4.3 System-Specific Interrupt Sources**

Source	SIPR Bit (5.7.3.1)	SPARC IRL
Video SIMM (MDI)	Bit <20>: VI	8
Parallel Port	Bit <8>: SBus 2	3
DBRI	Bit <11>: SBus 5	9
Keyboard Power-on Request	Bit <13>: SBus 7	13
Power-fail detect (AUXIO-1)	Bit <30>: ME	15

**A.II.5 Implementation ID numbers:**

(5.5.1) ECC IMPL = 0x1, ECC VER = 0x0 (Campus-2 EMC)

(5.6.1) VME IMPL = N/A

(7.2.4) IOMMU IMPL = 0x0, VER = 0x0 (MSI)

(9.2.1) Machine Type = 0x72

SCSI/Ethernet TYPE = 0xA (Campus-2 DMA2)

**A.II.6 I/O and DVMA Details****A.II.6.1 I/O Details and Options Supported**

Campus-2 supports an on-board LANCE Ethernet, Emulex ESP-100A (NCR 53C90A) SCSI, and a parallel port, all through an SBus ASIC called S4-DMA2. There is also a DBRI (Dual Basic Rate ISDN) interface supported as an SBus device. There is support for on-board floppy with an 82077 controller and AUXIO-0. The diagnostic LED register is not supported; instead a single LED is driven via the AUXIO-0 register. Video frame buffers are supported in the memory space with an MDI.

There is no support for a VMEbus interface or I/O cache. Campus-2 supports 512KB of EPROM. A power control interface is supported through AUXIO-1. A MIDI port is supported through the use of Serial Port B when AUXIO-0 bit<3> is '1'.

*Note that the ISDN audio device (See section 9.4) will be deleted for FCS if the DBRI is functional; will be decided prior to FCS.*

**A.II.6.1 LANCE Ethernet****A.II.6.1.1 Ethernet Port Address Map**

PA<35:00>	REGISTER	SIZE	TYPE
0xEF00C0000	LANCE Register Data Port	2-byte	RW
0xEF00C0002	LANCE Register Address Port	2-byte	RW
0xEF0400010 †	Ethernet Control/Status Register	4-byte	RW
0xEF0400014 †	Ethernet Test Control/Status Register	4-byte	RW
0xEF0400018 †	Ethernet Cache Valid Bits	4-byte	RW
0xEF040001C †	Ethernet Base Address Register	4-byte	RW

† Note: this page is shared with the SCSI port control access.

See the AM7990 LANCE data sheet for additional information.

**A.II.6.1.2 Ethernet Control/Status Register**

BIT	NAME	TYPE	MEANING
D<0>	E_INT	R	LANCE Interrupt is asserted
D<1>	ERR_INT	R	LANCE DVMA received an SBus ERR ack; clears on E_INVALID or E_RESET write with '1'
D<3:2>	E_DRAINING	R	'11' if E-cache draining, else '00'
D<4>	EN_INT	RW	Enables E_INT and ERR_INT to cause system Enet Int.
D<5>	E_INVALID	W	Marks all of E-cache as invalid. Reads as '0'
D<6>	SLAVE_ERR	RW1	Indicates wrong-sized access to LANCE. Write '1' to clear
D<7>	RESET	RW	When set, invalidates E-cache and resets LANCE
D<9:8>		R	rsvd
D<10>	DRAIN	RW	Forces drain of E-cache, clears when done
D<11>	DIS_W_DRAIN	RW	Disables drain of E-cache on LANCE descriptor writes
D<12>	DIS_R_DRAIN	RW	Disables drain of E-cache on LANCE slave reads
D<14:13>		R	rsvd
D<15>	ILACC	RW	modifies LANCE access timing
D<16>	DIS_BUF_WRT	RW	Disables buffering of slave writes to LANCE
D<17>	DIS_W_INVALID	RW	Disables E-cache invalidates upon LANCE slave writes
D<19:18>	BURST_SIZE	RW	set to 0x1 for use in Campus-2
D<20>	ALE/AS_	RW	1 = ALE, 0 = AS*; must match LANCE config
D<21>	LOOP_TEST	RW	Enables Ethernet loopback test
D<22>	TP_AUI	RW	With LOOP_TEST = 0, defines TP or AUI interface
D<27:23>		R	rsvd
D<31:28>	ID	R	Device ID = 0xA

**A.II.6.1.3 Ethernet Test Control/Status Register and Ethernet Cache Valid Bits Register**

These registers are provided for diagnostic purposes; for details see the DMA2 specification.

**A.II.6.1.4 Ethernet Base Address Register**

Bits <7:0> of this register provide bits <31:24> of the DVMA virtual address used by the LANCE; these bits are concatenated with the 24-bit address provided by the LANCE.

**A.II.6.2 Emulex ESP100A (NCR 53C90A) or FAS100A SCSI Port**

**A.II.6.2.1 SCSI Port Address Map**

ADDRESS	REGISTER	TYPE	
0xEF800000	Transfer Count Low	RW	<b>ESP registers (8-bit)</b>
0xEF800004	Transfer Count High	RW	
0xEF800008	FIFO Data	RW	
0xEF80000C	Command	RW	
0xEF800010	Status/Bus ID	RW	
0xEF800014	Interrupt Status/Timeout	RW	
0xEF800018	Sequence Step/Synch transfer period	RW	
0xEF80001C	FIFO Flags/Synch. offset	RW	
0xEF800020	Configuration	RW	
0xEF800024	Clock Conversion Factor	W	
0xEF800028	Reserved (test)	RW	
0xEF80002C	Configuration #2	RW	
0xEF800030	Configuration #3 (FAS100A only)	RW	
0xEF800034-0xEF8000FFF	Reserved	-	
0xEF400000 †	SCSI DVMA Control/Status Register	RW	
0xEF400004 †	SCSI DVMA Address Register	RW	
0xEF400008 †	SCSI DVMA Byte Count Register	RW	
0xEF40000C †	SCSI Test Control/Status Register	-	

† Note: this page is shared with the Ethernet port control access.

Refer to the ESP-100A (NCR53C90A) data sheet and the *Campus-II DMA2 Chip Specification, Sun P/N 950-xxxx-xx*, for a full programmer's model. To determine if this is an ESP100A or a FAS100A do the following probe: (1) reset the SCSI chip via the SCSI Control/Status Register. (2) configure this device to be target 7. (3) Attempt a RESELECT3 command to target 7. If this is an ESP100A the chip will issue an *Illegal\_Command* interrupt; if it is a FAS100A it will eventually issue a *Timeout* interrupt.

#### A.II.6.2.2 SCSI Control Register

BIT	NAME	TYPE	MEANING
D<0>	D_INT	R	ESP Interrupt asserted, or (TC = 1 and D_TCI_DIS = 0)
D<1>	ERR_INT	R	ESP DVMA received an SBus ERR ack; clears on E_INVALID or D_RESET write with '1'
D<3:2>	D_DRAINING	R	'11' if D-FIFO draining, else '00'
D<4>	EN_INT	RW	Enables D_INT and ERR_INT to cause system Enet Int.
D<5>	D_INVALID	W	Marks all of D-FIFO as invalid. Reads as '0'
D<6>	SLAVE_ERR	RW1	Indicates wrong-sized access to SCSI reg's. W/'1' to clear
D<7>	RESET	RW	When set, invalidates D-FIFO and resets ESP
D<8>	WRITE	RW	DMA direction for ESP DVMA; 1 = to memory
D<9>	EN_DMA	RW	Enables DMA requests from ESP if other conditions allow
D<12:10>		R	rsvd
D<13>	D_EN_CNT	RW	Enables internal byte counter, to be decremented on xfers
D<14>	D_TC	RW1	Set when byte count expires. W/1 to clear if D_EN_CNT=1
D<15>		R	rsvd
D<16>	DIS_CSR_DRN	RW	Disables drain of D-FIFO on writes to D-CSR
D<17>	DIS_ESP_DRN	RW	Disables drain of D-FIFO on writes to ESP registers
D<19:18>	BURST_SIZE	RW	Set to 0x1 for use in campus-2
D<20>	D_DIAG	RW	Disables drain/reset of D-FIFO on writes to D_ADDR
D<22:21>	SPEED	RW	Set to 0x2 for Campus-2 or to 0x1 if FAS100A is used †
D<23>	D_TCI_DIS	RW	Disables D_TC from generating an interrupt
D<24>	D_EN_NEXT	RW	Enables next address/count autoloading; requires D_EN_CNT=1
D<25>	D_DMA_ON	R	DMA2 can respond to ESP DMA; D<9> & (D<26> or D<27>) & ~(D<1>)
D<26>	D_A_LOADED	R	D_ADDR written or NEXT_ADDR loaded into D_ADDR
D<27>	D_NA_LOADED	R	NEXT_ADDR was written
D<31:28>	ID	R	Device ID = 0xA

† Campus-2 uses a NCR53C90A/ESP100A; the FAS100A may be specified for the FCS version

#### A.II.6.2.3 SCSI DVMA Address Register

BIT	NAME	TYPE	MEANING
D<31:0>	A<31:0>	read-write	Virtual address used in SCSI DVMA access

If the D\_EN\_NEXT bit in the SCSI CSR is set then a write to the SCSI DVMA address register will write to the NEXT\_ADDRESS register instead. If D\_EN\_NEXT is set when the byte count expires and the NEXT\_ADDRESS has been loaded since the last time this occurred, then NEXT\_ADDRESS is copied into the address register. See the DMA2 spec for further details.

#### A.II.6.2.4 SCSI DMA Count Register

BIT	NAME	TYPE	MEANING
D<31:24>	Reserved	R	Read as '0', writing has no effect
D<23:0>	COUNT	RW	SCSI DMA transfer count

This counter is decremented each time a byte is transferred between the DMA2 and the ESP in a DMA cycle. When the counter reaches '0' the D\_TC bit in the SCSI CSR is set. Loading with '0' causes the maximum transfer (2\*\*24 bytes). If the D\_EN\_NEXT bit in the SCSI CSR is set then a write to the SCSI DMA count register will write to the NEXT\_COUNT register instead. If D\_EN\_NEXT is set when the byte count expires and the NEXT\_COUNT has been loaded since the last time this occurred, then NEXT\_COUNT is copied into the count register. See the DMA2 spec for further details.

### A.II.6.3 Parallel Port

The bidirectional 8-bit parallel port is a highly programmable interface that supports a wide variety of 'Centronics' interface specifications with a choice of DVMA or programmed I/O. Both chained and unchained operations are supported. Another operating mode can be used to accelerate clearing of memory. For a complete detailed programmer's models see the *Campus-II DMA2 Chip Specification*.

#### A.II.6.3.1 Parallel Port Address Map

PA<35:00>	REGISTER	SIZE	TYPE
0xEF480000	PP DMA Control/Status Register	4-byte	RW
0xEF480004	PP Address Register	4-byte	RW
0xEF480008	PP Byte Count Register	4-byte	RW
0xEF48000C	PP Test Control/Status Register	4-byte	RW
0xEF480010	PP Hardware Configuration Register	2-byte	RW
0xEF480012	PP Operation Configuration Register	2-byte	RW
0xEF480014	PP Parallel Data Register	1-byte	RW
0xEF480015	PP Transfer Control Register	1-byte	RW
0xEF480016	PP Control Output Register	1-byte	RW
0xEF480017	PP Status Input Register	1-byte	RW
0xEF480018	PP Interrupt Control Register	2-byte	RW

#### A.II.6.3.2 Parallel Port Control/Status Register

BIT	NAME	TYPE	MEANING
D<0>	P_INT	R	PP_DMA or PP Ctrl interrupt, or (TC=1 and P_TCI_DIS=0)
D<1>	ERR_INT	R	PP DVMA received an SBus ERR ack; clears on E_INVALID or P_RESET write with '1'
D<3:2>	P_DRAINING	R	'11' if P-FIFO draining, else '00'
D<4>	EN_INT	RW	Enables P_INT and ERR_INT to cause system Enet Int.
D<5>	P_INVALID	W	Marks all of P-FIFO as invalid. Reads as '0'
D<6>	SLAVE_ERR	RW1	Indicates wrong-sized access to SCSI reg's. W/'1' to clear
D<7>	RESET	RW	When set, invalidates P-FIFO and resets ESP
D<8>	WRITE	RW	DMA direction for PP DVMA; 1 = to memory
D<9>	EN_DMA	RW	Enables DMA transfers to/from PP
D<12:10>		R	rsvd
D<13>	P_EN_CNT	RW	Enables internal byte counter, to be decremented on xfers
D<14>	P_TC	RW1	Set when byte count expires. W/1 to clear if P_EN_NEXT=1
D<17:15>		R	rsvd
D<19:18>	BURST_SIZE	RW	Set to 0x1 for use in campus-2
D<20>	P_DIAG	RW	Disables drain/reset of P-FIFO on writes to P_ADDR
D<22:21>		R	rsvd
D<23>	P_TCI_DIS	RW	Disables P_TC from generating an interrupt
D<24>	P_EN_NEXT	RW	Enables next address/count autoloading; requires P_EN_CNT=1
D<25>	P_DMA_ON	R	DMA is not disabled due to any hardware/software reason
D<26>	P_A_LOADED	R	Address and byte count are valid during chained transfer
D<27>	P_NA_LOADED	R	NEXT_ADDR and NEXT_COUNT written but not used
D<31:28>	ID	R	Device ID = 0xA

**A.II.6.3.3 Parallel Port DVMA Address Register**

BIT	NAME	TYPE	MEANING
D<31:0>	A<31:0>	read-write	Virtual address used in PP DVMA access

If the P\_EN\_NEXT bit in the PP CSR is set then a write to the PP DVMA address register will write to the NEXT\_ADDRESS register instead. If P\_EN\_NEXT is set when the byte count expires and the NEXT\_ADDRESS has been loaded since the last time this occurred, then NEXT\_ADDRESS is copied into the address register. See the DMA2 spec for further details.

**A.II.6.3.4 Parallel Port DMA Byte Count Register**

BIT	NAME	TYPE	MEANING
D<31:24>	Reserved COUNT	R	Read as '0', writing has no effect
D<23:0>		RW	PP DMA transfer count

This counter is decremented each time a byte is transferred between the DMA2 and the device connected to the parallel port in a DMA cycle. When the counter reaches '0' the P\_TC bit in the PP CSR is set. Loading with '0' causes the maximum transfer (2\*\*24 bytes). If the P\_EN\_NEXT bit in the PP CSR is set then a write to the PP DMA count register will write to the NEXT\_COUNT register instead. If P\_EN\_NEXT is set when the byte count expires and the NEXT\_COUNT has been loaded since the last time this occurred, then NEXT\_COUNT is copied into the count register. See the DMA2 spec for further details.

**A.II.6.3.5 Parallel Port Test Control/Status Register**

Provided for diagnostic purposes; see the DMA2 specification.

**A.II.6.3.6 Parallel Port Hardware Configuration Register**

15	14	8	7	6	0
TEST	DSW		rsvd	DSS	

Field	Description	Type
TEST	Allows buried registers to be accessed	RW
DSW	Data strobe width, in SBus clocks	RW
DSS	Data setup before data strobe, in SBus clocks	RW

**A.II.6.3.7 Parallel Port Operation Configuration Register**

15	14	13	12	11	10	9	8	7	6	4	3	2	0
MEM	DATA	DSEL	BDS	ADS	DIAG	BOP	AOP	SRST	rsvd	IDLE	rsvd		

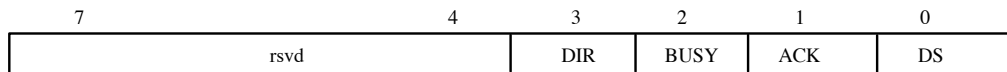
Field	Description	Type
MEM	Enable memory clear operation	RW
DATA	Data sourced will be all 1's or all 0's	RW
DSEL	Enables data strobe to be bidirectional	RW
BDS	Enables BUSY to be bidirectional	RW
ADS	Enables ACKNOWLEDGE to be bidirectional	RW
DIAG	Enables diagnostic mode	RW
BOP	BUSY operation 1 = use BSY	RW
AOP	ACKNOWLEDGE operation; 1 = use ACK	RW
SRST	Resets the parallel port; must be released by SW	RW
IDLE	Indicates that PP state machines are idle	R

**A.II.6.3.8 Parallel Port Parallel Data Register**

This 8-bit read/write register is used for programmed I/O when P\_DMA\_EN = 0; if the DIR bit of the Transfer Control Register is '0' this register is written to send data, when it is '1' it is used to read the data latched with the last data strobe.

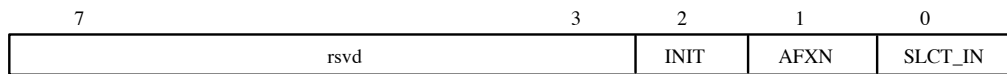
Diagnostic loopback is possible by doing a single DMA cycle followed by a single PIO cycle.

**A.II.6.3.9 Parallel Port Transfer Control Register**



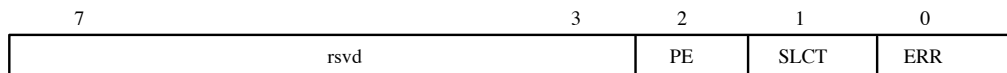
Field	Description	Type
<b>DIR</b>	<b>1 = write to external device, 0 = read from external device</b>	<b>RW</b>
<b>BUSY</b>	<b>Read shows value of <i>busy</i> pin; write with BDS = 1 and DIR = 1 drives the value onto <i>busy</i>.</b>	<b>RW</b>
<b>ACK</b>	<b>Read shows value of <i>ack</i> pin; write with ADS = 1 and DIR = 1 drives the value onto <i>ack</i>.</b>	<b>RW</b>
<b>DS</b>	<b>Read shows value of <i>data_strobe</i>; write with DSEL = 0 or (DSEL = 1 and DIR = 0) drives the value onto <i>data_strobe</i></b>	<b>RW</b>

**A.II.6.3.10 Parallel Port Control Output Register**



Field	Description	Type
<b>INIT</b>	<b>Initialize: driven onto the <i>init</i> pin</b>	<b>RW</b>
<b>AFXN</b>	<b>Auto Feed: driven onto the <i>afxn</i> pin</b>	<b>RW</b>
<b>SLCT_IN</b>	<b>Select In: driven onto the <i>slct_in</i> pin</b>	<b>RW</b>

**A.II.6.3.11 Parallel Port Status Input Register**



Field	Description	Type
<b>PE</b>	<b>Paper Empty: reads the <i>pe</i> pin</b>	<b>RW</b>
<b>SLCT</b>	<b>Select: reads the <i>slct</i> pin</b>	<b>RW</b>
<b>ERR</b>	<b>Error: reads the <i>err</i> pin</b>	<b>RW</b>



**A.II.6.3.12 Parallel Port Interrupt Control Register (Interrupts on SBus L2)**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
DI	AI	BI	PI	SI	EI	DIE	AIE	BP	BIE	PP	PIE	SP	SIE	EIP	EIE

Field	Description	Type
DI	Data Strobe Interrupt Pending; write '1' to clear	RW1
AI	Acknowledge Interrupt Pending; write '1' to clear	RW1
BI	Busy Interrupt Pending; write '1' to clear	RW1
PI	Paper Empty Interrupt Pending; write '1' to clear	RW1
SI	Select Interrupt Pending; write '1' to clear	RW1
EI	Error Interrupt Pending; write '1' to clear	RW1
DIE	Enable DI on rising edge of data_strobe	RW
AIE	Enable AI on rising edge of ack	RW
BP	Busy interrupt polarity; 1 = rising edge, 0 = falling edge	RW
BIE	Enable BI on edge of busy selected by BP	RW
PP	Paper Empty polarity; 1 = rising edge, 0 = falling edge	RW
PIE	Enable PI on edge of paper_empty selected by PP	RW
SP	Select interrupt polarity; 1 = rising edge, 0 = falling edge	RW
SIE	Enable SI on edge of select selected by SP	RW
EP	Error interrupt polarity; 1 = rising edge, 0 = falling edge	RW
EIE	Enable EI on edge of select selected by EP	RW

**A.II.6.4 DBRI Interface**

**A.II.6.4.1 DBRI Port Description**

Campus-2 provides a Dual Basic-Rate ISDN (DBRI) interface. This DVMA device resides on the SBus, and shares the address space of the on-board slot with the DMA2 chip. Connections for this interface are an ISDN Terminal Endpoint (TE) and Network Termination (NT) plus a bus that uses the AT&T Concentration Highway Interface (CHI) for communication with an external device.

TE and NT together provide a standard connection for an ISDN layer-1 4-wire interface. The CHI interface allows for connection to an external device; the planned device is a CODEC interface in a multimedia peripheral called SpeakerBox. CHI is accessed through the multi-function audio/AUI connector with a break-out pigtail cable. It is intended to connect to the Speakerbox project from the MMP group.

**A.II.6.4.2 DBRI Port Address Map**

PA<35:00>	REGISTER	SIZE	TYPE
0xEF801000	DBRI Control/Status Register (REG0)	4-byte	RW
0xEF801004	DBRI Mode and Interrupt Reg (REG1)	4-byte	RW
0xEF801008	DBRI Parallel I/O Register (REG2)	4-byte	RW
0xEF80100C	DBRI Test Mode Register (REG3)	4-byte	RW
0xEF801020	DBRI Program Counter Reg (REG8)	4-byte	RW
0xEF801024	DBRI Interrupt Queue Pointer Reg (REG9)	4-byte	RW

**A.II.7 Bugs/Features**

MID Register: A bug in the first implementation of the MID register makes this function unusable in this system. The description of the MID Register in section 5.4.3 describes general workarounds for this bug.

### A.II.8 Board Partition

There are 7 ASICs in the Campus-2 design. The MSI (MBus/SBus Interface) implements the M-to-S and S-to-M functions, including the IOMMU, write buffers in both directions, the SBus arbiter, and the MBus arbiter. The M-to-S asynchronous error registers and the arbiter enable register reside in this chip, along with all SBus-related and IOMMU related registers. This chip was developed as part of Galaxy (A.I)

The EMC (ECC Memory Controller) interfaces the MBus to main memory. It contains ping-pong write buffers, ECC generation and check logic, and the ECC error registers. It also generates the controls to the memory SIMM's. For VRAM SIMMs it provides the extra address controls and timing.

The SEC (SBus/EBus Controller) implements the Sun-4M multiprocessor interrupt logic, and interfaces the system to the 8-bit slave I/O on the E-bus (*'E' stands for 'Eight-bit'. Well, what did you expect?'*). The SEC is accessed through the SBus, although it is not an SBus device. The SEC uses a special multiplexed mode to receive address information from the MSI on the S\_D bus, and does not connect to the S\_A bus. It also contains the system status and control register. The SEC was developed as part of the Galaxy program; some Galaxy pins that are not used on Campus2 (VME interrupts, P2\_IRL and P3\_IRL) are remapped via the C2\_MODE pin into interface to a floppy controller, a video controller, and a generic 8-bit slave device.

The DMA2 is an SBus device that interfaces the SBus to an on-board LANCE Ethernet, to an Emulex ESP100A SCSI interface, and to a highly programmable parallel port.. This same ASIC can be used on an SBus expansion card.

The Campus-2 Clock chip generates system clocks for MBus, SBus, and the serial port devices; it also serves as the board controller for JTAG scan testing.

MDI is a frame buffer controller that is used on the VRAM SIMM's; it is described in A.II.4.2.3.

The VBC ASIC on the VRAM SIMM controls refresh of the VRAM. It is not visible to software.

### A.II.9 Official Product Designations

*TBD.*

**B.I: Module Notes: Viking/NE**

*Based on the Viking Microprocessor User Documentation, rev 2.00 11/01/90. See that document for details.*

- B.I.1       Module Overview
- B.I.2       Cache Details
  - B.I.2.1               Instruction Cache Details
  - B.I.2.2               Data Cache Details
  - B.I.2.3               External Cache Details
- B.I.3       Cache Coherence, Store Buffers, and Memory Models
  - B.I.3.1               Cache Coherence
  - B.I.3.2               Store Buffers
  - B.I.3.3               Memory Models
- B.I.4       Module Registers: differences from core (section 4)
- B.I.5       MMU Details
- B.I.6       ASI's Implemented
- B.I.7       Module Control Space Address Map
- B.I.8       IU PSR Number
- B.I.9       Module-Specific Quirks

**B.I.1 Module Overview**

The Viking/NE module consists of a chip developed jointly by Texas Instruments and Sun. The processor is a superscalar SPARC processor and cache system. The module contains the Viking chip (TMS390Z50) with internal IU, floating-point, I- and D-caches, and MMU. The module runs synchronous with the 40 MHz MBus clock.

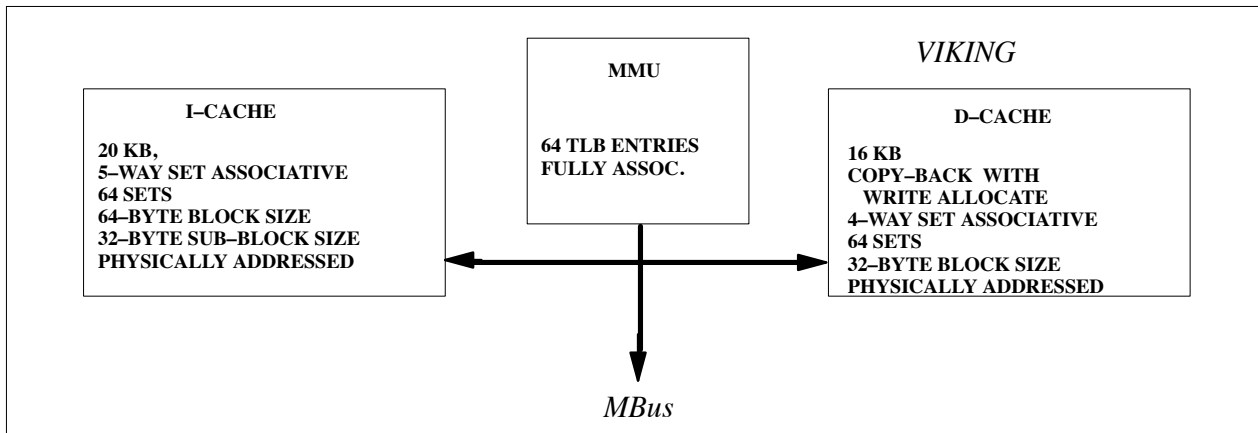
A physical Viking/NE module could have one or two Viking/NE logical modules on it. Both uni- and dual-processor modules may be built. (*Official name(s) TBD*).

A similar module with an External Cache (E-\$) consists of a Viking chip plus an external cache controller MXCC and 8 pipelined 128K x 9 SRAM chips (some versions may use 128K x8). That module is called Viking/E\$, and it is defined in appendix B.III. The module type is identified by examining bit 11 of the Module Control Register (MCR). If the bit is '1' then this is the correct model; if the bit is a '0' then the Viking/E\$ appendix applies.

The module in appendix B.III will be considered the standard module; wherever possible this appendix will identify differences in Viking/NE.

**B.I.2 Cache Details**

The Viking/NE system utilizes a Harvard architecture with a 20KB physical-address set-associative cache for instruction access, and a 16KB physical-address set-associative cache for data access. The caches participate in the MBus level-2 coherence protocols. Snooping is implemented via the D-\$ and E-\$ directories. The data cache can be pin-programmed to be either a write-through cache with a no-write-allocate policy or a copy-back cache with write-allocate. The latter is required for Viking/NE.



**B.I.2.1 Instruction Cache Details**

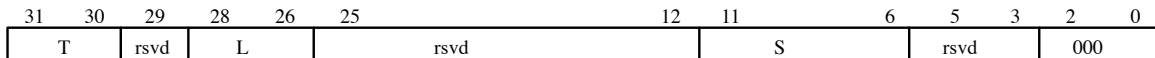
The I-cache is 5-way set associative. It consists of 64 sets with a block-size of 64 bytes, and each block is divided into two sub-blocks of 32 bytes each. Each set has an *Stag* (Set Tag) associated with it, and each line within the set has a *Ptag* (Physical Tag) associated with it.

The *Stag* contains a lock bit for each line within the set to allow selective locking of cache lines, with the exception of line 0x0; writes to that bit are ignored. Also visible in the *Stag* are the MRU (most-recently used) limited history bits used to implement the set replacement algorithm. Details of the algorithm are available in the Viking User's Guide. All bits can be read and written for diagnostic and locking purposes.

The Ptag for each block in the I-\$ directory contains the *physical tag* PA<35:12> for the block plus a *valid* bit for each sub-block. The I-\$ snoops write activity and will invalidate accordingly. If code modifies instructions, a SPARC *flush* must be executed to ensure consistency; the *flush* will synchronize on the completion of all pending writes, and will also flush the instruction prefetch buffer. The I-\$ can be initialized with a *flash-clear* mechanism. This is required after power-up.

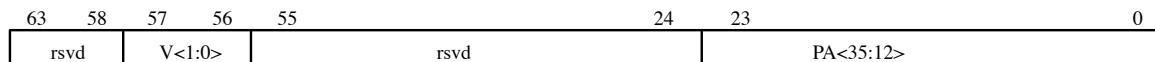
The state of I-cache tags and data are unaffected by Watchdog or system reset. The I-cache is enabled by the IE bit in the MCR.

**B.I.2.1.1 Instruction Cache Tag Address Format (ASI = 0xC)**



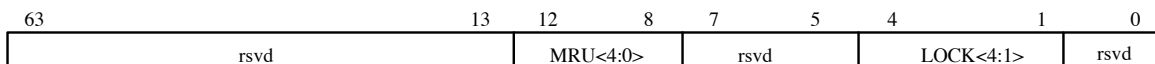
Field	Description
T	Type of tag; 1 = Set Tag (stag), 2 = Physical Tag (ptag), 0 = 3 = reserved
L	Line within set (0-4); (5-7) = reserved
S	Set (1 of 64)
rsvd	reserved: these bits are ignored

**B.I.2.1.2 Instruction Cache Ptag Format (64-bit access only)**



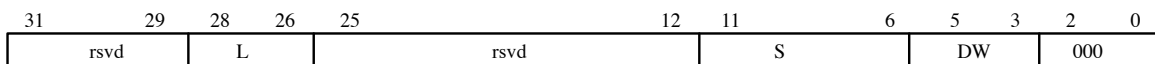
Field	Description	Type
V<1:0>	Valid bit for the 32-byte sub-blocks; V<1> corresponds to the high-order sub-block (A<4> = 1) and V<0> to the low-order.	RW
PA<35:12>	Physical tag address bits	RW
rsvd	reserved: read as '0'	R

**B.I.2.1.3 Instruction Cache Stag Format (64-bit access only)**



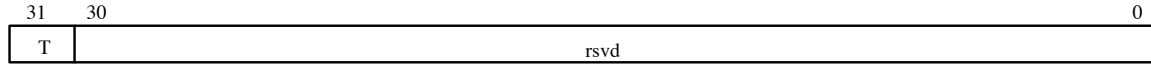
Field	Description	Type
MRU<4:0>	Access history bits used in the partial-LRU algorithm	RW
LOCK<4:1>	Lock bits for blocks <4:1> within the set.	RW
rsvd	reserved: read as '0'	R

**B.I.2.1.4 Instruction Cache Data Address Format (ASI = 0xD)**



Field	Description
L	Line within set (0-4); (5-7) = reserved
S	Set (1 of 64)
DW	Double-word within line
rsvd	reserved: these bits are ignored

**B.I.2.1.5 Instruction Cache Flash Clear (ASI = 0x36)**



Field	Description	Type
T	Type: 0 = clear all Valid and MRU bits in PTags and Stags. 1 = clear all Lock bits in Stags.	W
rsvd	reserved: ignored	W

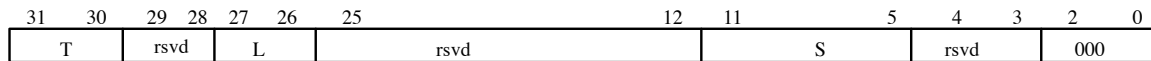
**B.I.2.2 Data Cache Details**

The D-cache is 4-way set associative. It consists of 128 sets with a block-size of 32 bytes; there is no sub-blocking. Each set has an *Stag* (Set Tag) associated with it, and each line within the set has a *Ptag* (Physical Tag) associated with it. In the Viking/NE module the D-\$ is configured as a copy-back cache with a write-allocate policy.

The *Stag* contains a lock bit for each line within the set to allow selective locking of cache lines, with the exception of line 0x0. Also visible in the *Stag* are the MRU (most-recently used) limited history bits used to implement the set replacement algorithm. Details of the algorithm are available in the Viking User's Guide. All bits can be read and written for diagnostic and locking purposes.

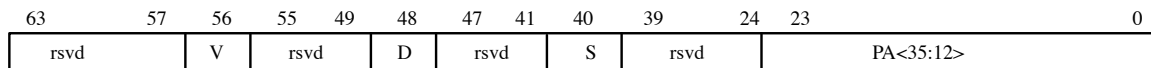
The *Ptag* for each block in the D-\$ directory contains the *physical tag* PA<35:12> plus a *valid*, *dirty*, and *shared* bit for the block. The D-\$ can be initialized with a *flash-clear* mechanism. This is required after power-up. The state of D-cache tags and data are unaffected by Watchdog or system reset.

**B.I.2.2.1 Data Cache Tag Address Format (ASI = 0xE)**



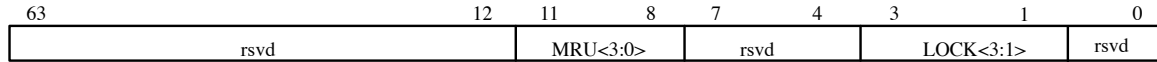
Field	Description
T	Type of tag; 1 = Set Tag (stag), 2 = Physical Tag (ptag), 0 = 3 = reserved
L	Line within set (0-3)
S	Set (1 of 128)
rsvd	reserved: these bits are ignored

**B.I.2.2.2 Data Cache Ptag Format**



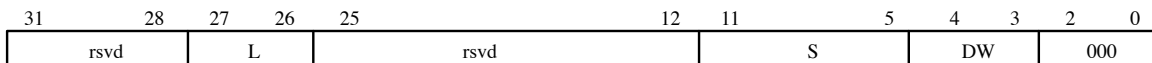
Field	Description	Type
V	Valid bit for the 32-byte block	RW
D	Dirty	RW
S	Shared	RW
PA<35:12>	Physical tag address bits	RW
rsvd	reserved: read as '0'	R

**B.I.2.2.3 Data Cache Stag Format**



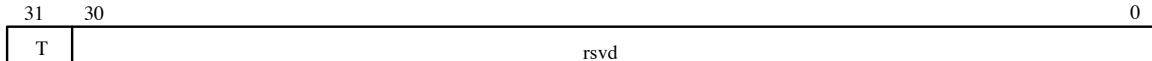
Field	Description	Type
MRU<3:0>	Access history bits used in the partial-LRU algorithm Lock bits for blocks within the set. reserved: read as '0'	RW
LOCK<3:1>		RW
rsvd		R

**B.I.2.2.4 Data Cache Data Address Format (ASI = 0xF)**



Field	Description
L	Line within set (0-3) Set (1 of 128) Double-word within line reserved: these bits are ignored
S	
DW	
rsvd	

**B.I.2.2.5 Data Cache Flash Clear (ASI = 0x37)**



Field	Description	Type
T	Type: 0 = clear all Valid and MRU bits in PTags and Stags. 1 = clear all Lock bits in Stags.	W
rsvd	reserved: ignored	W

**B.I.2.2.6 D-Cache Data Prefetching**

In Viking/NE mode data prefetch is not supported, and the PF bit in the MCR is ignored. **This is different from Viking/E\$.**

**B.I.3 Cache Coherence, Store Buffers, and Memory Models**

**B.I.3.1 Cache Coherence**

**B.I.3.1.1 Cache Flushing**

Since Viking caches are physically addressed the contents of cache lines always corresponds to a backing store in physical memory. For this reason no flush mechanism is supported; there is never a need to dereference the contents of any cache line.

For diagnostic purposes, flushes can be forced by accessing a different physical address that maps to the same cache line, i.e. a displacement flush. A D-\$ set can be flushed by reading in 4 cacheable addresses that map to the same set, i.e. modulo [4KB], or by using the D-\$ flash-clear mechanism. Similarly the entire I-\$ can be flushed with the flash-clear operation. Flash-clear invalidates all tags; selective invalidaton can be done by clearing specific valid bits through the diagnostic ASI path.

### B.I.3.1.2 Cache Aliasing

Since all caches in this system are physically addressed there is no aliasing rule needed.

### B.I.3.1.3 Cache Snooping

The D-\$ snoops all coherent MBus transactions based on the physical address, and will implement the standard MOESI model of cache coherence. Snoop hits cause the MBus level-2 signals MSH\* and MIH\* to be asserted in cycle A+3 if appropriate.

The I-\$ snoops all transactions that go by on the MBus, including those initiated by this processor's D-\$ side. A snoop hit on any CI, CRI, or CWI will cause the I-\$ to invalidate that line. A snoop hit on a CR will cause the I-\$ to assert MSH\* in cycle A+3. **The I-\$ sees all stores as they leave the IU pipeline. The SPARC *flush* instruction merely flushes the IU pipeline in Viking/NE.**

### B.I.3.2 Store Buffers

#### B.I.3.2.1 Store Buffer Operation

Viking implements a store buffer which behaves as a FIFO; internally it is configured as a fully-associative cache of 8 double-words. In Viking/NE the store buffer is used to hold copybacks of modified cache data, as well as non-cached stores.

The store buffer is automatically drained upon a context switch. If a store buffer exception occurs as a result of the copyout, the trap will be reported to the IU *before* the *sta* to the context register completes, so the contents of the CTX register will reflect the context that owns the store buffer trap. Upon resuming execution after the trap, the store to CTX will complete.

The store buffer is enabled by the SB bit in the MCR.

#### B.I.3.2.2 Store Buffer Exceptions

When a fault occurs on a buffered store, it is reported to the IU as a `data_store_error` trap (priority 2, type 0x2b). In order for the error to be reported the NF bit in the MCR must be 0x0 and traps must be enabled. Upon the detection of a store buffer fault (due to a report of an error from the MBus) the store buffer is frozen, the SB bit in the MCR is cleared, and the store buffer will retain all pending stores including the faulting one. A buffer copyout is *not* initiated. All subsequent store accesses are synchronous and behave as if the store buffer is disabled. Nominally these stores belong to the trap handler, and so belong to a different, unrelated thread of execution. Note that this is a synchronous trap issued for an asynchronous event, similar to an interrupt.

The trap handler can retry the store operation by obtaining the data, address, size, and cacheability information from the store buffer using diagnostic access. The store can be recreated by using an ASI bypass with *sta*; the AC bit in the MCR should be set to match the C bit in the store buffer tag. The trap handler should set the NF bit prior to retrying the store. After the *sta* the handler can examine the SFSR to determine the error status. After the fault status is determined the OS can decide what action to take.

Alternatively the trap handler can re-issue the faulted store by simply re-enabling the store buffer. When a store buffer exception occurs, the store buffer pointers retain their present value. Setting NF=0 will prevent another store buffer trap from being taken if the store sees another exception. In this case, the store buffer will still turn off, even with NF=0. However, the store buffer error remains pending, and the IU-pipe doesn't see it until NF=1. A bit in the store buffer control register indicates the presence of a pending (unacknowledged) store buffer error, or the code could check if the store buffer is still enabled after the store.

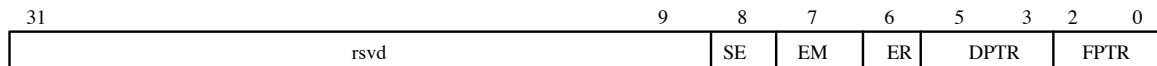


Store order can be maintained even in the presence of an error; the trap handler simply retries all pending stores in the order they were placed in the buffer.

After handling an error, the trap handler can return the store buffer to operation by clearing all valid bits, setting the *drain* pointer equal to the *fill* pointer in the Store Buffer Control Register, and then setting the SB bit in the MCR to 0x1.

**A quirk in the Viking/NE module is that a store buffer error will cause both a type 0x2b trap (priority 2) and a broadcast level-15 module\_error interrupt. The trap will be taken first, but afterwards there will be a pending level-15 broadcast interrupt waiting for this module.**

**B.I.3.2.3 Store Buffer Control Register (ASI = 0x32)**

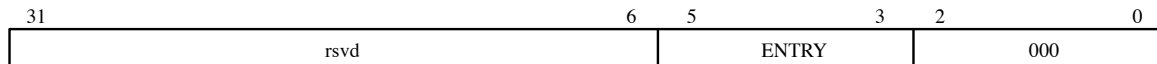


Field	Description	TYPE
SE	Store Buffer Enabled: shadow of bit in MCR	R
EM	Store Buffer Empty: 1 = Empty. Set to '1' at reset.	R
ER	Store Error Pending: set when a store buffer error occurs when traps are disabled. Cleared when the trap is taken.	R
DPTR	Drain Pointer: indicates the buffer entry that is at the head of the queue	RW
FPTR	Fill Pointer: indicates the next entry that will be written to.	RW

The store buffer is full if the fill pointer equals the drain pointer and there are valid entries. The buffer is empty if the two pointers are equal and there are no valid entries. The pointers increment modulo 8.

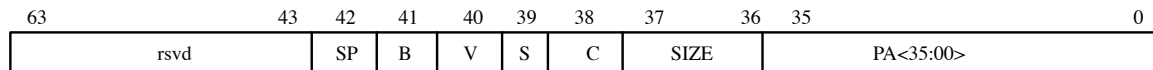
**B.I.3.2.4 Diagnostic Access to the Store Buffer**

**B.I.3.2.4.1 Store Buffer Tag Address Format (ASI = 0x30)**

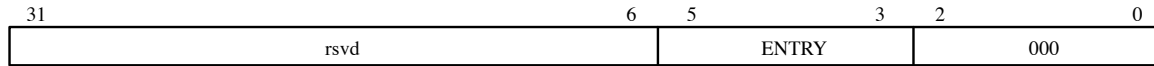


Field	Description
ENTRY rsvd	Store buffer entry number reserved: these bits are ignored

**B.I.3.2.4.2 Store Buffer Tag Format**



Field	Description
SP	Store Barrier Pointer: Used in PSO to mark the synchronization point
B	Burst Mode Access: Indicates that the next entry in the buffer corresponds to the next address, and thus can be issued in a burst
V	Valid entry
S	Supervisor: Indicates that the store was issued by a supervisor thread
C	Cacheable: Indicates that this store is a cacheable access
SIZE	Size of access: 00 = byte, 01 = half-word, 10 = word, 11 = double-word
PA<35:00> rsvd	Physical address of the store. Must be size-aligned or an error will occur reserved: these bits are ignored

**B.I.3.2.4.3 Store Buffer Data Address Format (ASI = 0x31)**

Field	Description
ENTRY rsvd	Store buffer entry number (double-word access) reserved: these bits are ignored

**B.I.3.3 Memory Models**

Viking is capable of operating in three modes of ordering: *strong sequential ordering*, *Total Store Ordering (TSO)*, and *Partial Store Ordering (PSO)*. The first is the standard model of data ordering in which all loads and stores complete in order of issue. The latter two models are those defined in the SPARC Architecture Manual version 8.

Strong ordering is achieved by disabling the internal store buffer. This is done by setting the SB bit in the MCR to 0x0. TSO is enabled by setting the SB bit to 0x1 and the PSO bit in the MCR to 0x0. PSO is enabled by setting both SB and PSO to 0x1. TSO is the nominal memory model in Sun-4M and in SPARC V.8. PSO requires explicit synchronization of stores; stores are guaranteed to be ordered only with respect to the SPARC synchronization instruction *stbar*.

**B.I.4 Module Registers: differences from core (section 4)****B.I.4.1 Module Control Register: differences from core (section 4.1)**

Bit <7>: PSO, rw. 1 = PSO mode, 0 = TSO mode. **Always set to '0' in Sun-4M systems.**

Bit <8>: DE, rw. Data-cache enable

Bit <9>: IE, rw. Instruction-cache enable.

Bit <10>: SB, rw. Store buffer enable.

Bit <11>: MB, ro. 0 = E-cache mode, this is a Viking/E\$ module and the D-\$ is write-through. 1 = MBus mode, this is a Viking/NE module and the D-\$ is copy-back mode (if '0' then appendix B.III applies).

Bit <12>: PE, rw. Parity check enable; **Always set to '0' in Viking/NE systems**

Bit <13>: BT, rw. Boot mode. 0 = normal operation, 1 = boot mode. This bit functions exactly like bit <14> in the generic register specification in section 4.1.

Bit <14>: SE, rw. Snoop Enable: when '1' snooping is enabled for the I-\$ and D-\$.

Bit <15>: AC, rw. Alternate cacheable. When the MMU is disabled or when the PTE is not used for translation (i.e. for MMU bypass accesses) this bit provides the cacheability status of all transactions **with the exception of boot-mode instruction fetches, which are never cacheable.** This bit affects both the internal I-\$ and D-\$. This functions differently from some modules, which will use the AC bit as an advisory for the MBus address phase but will not cache these references. In Sun-4M machines care must be taken to not accidentally attempt cacheable accesses to non-memory devices or an error acknowledge will result.

Bit <16>: TC, rw. Table-walk cacheable in E-\$. **Always set to '0' in Viking/NE systems**

Bit <18>: PF, rw. Data Prefetch enable. **Ignored in Viking/NE.**

Bits <27:24>: SRMMU VER = 0x0, ro. Mask revisions will show only in the IU PSR.

Bits <31:28>: SRMMU IMPL = 0x4, (Texas Instruments), ro

**B.I.4.2 Context Table Pointer Register: differences from core (section 4.2)**

The Context Table Pointer Register has bits CTPR <31:8> = CTP <35:12>, and bits CTPR <7:0> are reserved. Depending on the number of contexts supported the CTP may have any number of bits from <18:12> zeroed; the context table must be size aligned. CTP <17:12> is the logical **OR** of CTPR <13:8> and CTX <15:10>. Thus if there are 10 context bits, CTPR<31:8> represent CTP<35:12> and the context register bits CTX<9:0> represent CTP<11:2>, with CTP<1:0> = 00; the context table would have to be 4K aligned. At the far end of the spectrum, if there are 16 context bits then CTPR<31:14> represent CTP<35:18>, CTX<15:0> = CTP<17:2>, and again CTP<1:0> = 00; then the context table would have to be 256K aligned. Any size in between is also supported. At a minimum the table must be 4K aligned.

**B.I.4.3 Context Register: differences from core (section 4.3)**

N = 15, thus up to 64K contexts are supported.

**B.I.4.4 Synchronous Fault Status Register: differences from core (section 4.4.1)**

Bit <12>: UC: Uncorrectable Error. **In addition to uncorrectable errors, Viking/NE will post this bit if it receives a RETRY acknowledge on the MBus.** This should never occur in a Sun-4M system.

Bit <13>: VMP: Viking Master Parity Error. **Not used in Viking/NE modules.**

Bit <14>: P: Parity Error. **Not used in Viking/NE modules.**

Bit <15>: SB: Store Buffer Error. Indicates that a store buffer error has been detected (see section B.I.3.2.2). **In Viking/NE assertion of this bit causes assertion of the module\_error interrupt pin; it will deassert when this register is read.**

Bit <16>: CS: Control Space Access Error. Set if a *lda*, *sta*, or *swapa* returns an access error, except for bus error on ASI's 0x8 - 0xB and 0x20 - 0x2F or for bus error on MMU probe operations. Normally these errors will occur if there is an invalid ASI space, an incorrect size of access, or an invalid address within a valid ASI space. MFAR will hold the correct address in case of a CS error, although the ASI is not captured anywhere.

Bit <17>: EM: Error Mode Reset Taken. When set this indicates that the processor has taken a watchdog reset. **In Viking/NE this substitutes for the WD bit in the reset register, which is not supported. Assertion of this bit causes assertion of the module\_error interrupt pin; it will deassert when this register is read.**

**B.I.4.5 Synchronous Fault Address Register: differences from core (section 4.4.2)**

As allowed in the SRMMU specification this register will never hold the address of an instruction fault; that information will be found in the saved PC/NPC. A special diagnostic path allows for this register to be written as well as read at ASI = 0x4, VA = 0x00001400.

**B.I.4.6 Asynchronous Fault Status and Address Registers: differences from core (section 4.5)**

Viking does not support an AFSR/AFAR, even though it contains write buffers which allows write operations to complete asynchronous to the instruction flow. Any error reported asynchronously will be captured in MXCC registers. See B.I.3.2.2 'Store Buffer Exceptions' for more information.

**B.I.4.6 Reset Register: differences from core (section 4.6)**

There is no Reset Register internal to Viking. **The Watchdog Reset status is found in SFSR bit <17>.**

Assertion of WD will cause assertion of the Module Error output. **This interrupt is cleared by reading the SFSR.** Snooping is maintained during either an SI or a WD reset.

**B.I.4.7 MBus Port Address Register Register: differences from core (section 4.6)**



Field	Description	Type
MDEV	MBus device number = 0x0 (Viking/NE)	R
MREV	Device revision number = 0x0	R
MVEND	MBus vendor number = 0x4 (Texas Instruments)	R
rsvd	reserved, read as '0'	R

The MBus Port Address Register can only be accessed via module control space.

**B.I.5 MMU Details**

The MMU contains 64 fully-associative TLB entries with a limited-history LRU replacement algorithm. **The page tables are not cacheable in Viking/NE.** The Viking MMU implements the full set of PROBE types.

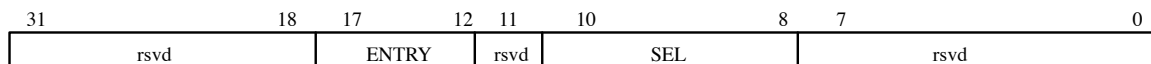
The limited-history replacement algorithm first sweeps TLB entries sequentially until all TLB's have the valid bit set. From that point the next access to any TLB will set its 'used' bit; replacement cycles will take the first TLB that is not 'used'. At the time that all TLB entries have the 'used' bit set, all 'used' bits except the last one to be set are cleared, and all history is lost. The demap-all operation will clear all 'used' and 'valid' bits.

The root pointer and the most recently used level-2 PTP are also cached in the MMU. A context switch will invalidate the cached root pointer. Upon the first miss after a context switch, an extra level of table-walk is supported in order to fetch that context's root pointer. The level-2 PTP cache is invalidated upon writes to the context register or the context table pointer register, or upon table-walks that do not use that PTP (in which case a new level-2 PTP is obtained for the cache). This cached PTP is used only for table-walks, M-bit updates, and probe-entire operations. If level-2 is a PTE then it is not cached.

A table walk is *not* atomic on the MBus. Updates to the Modified and Referenced bits will guarantee correctness as follows: (1) if this is the first access to the page and it is not a store, the MMU will do an atomic swap operation to set the 'R' bit; if the swap return 'R' and 'M' both set (by another processor) then another swap is performed to set both 'R' and 'M'; (2) if this is a store access and the 'M' bit is not set in the PTE, a simple store is done with both 'R' and 'M' set.

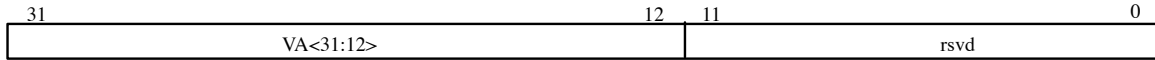
Because hardware may be iteratively updating the PTE when software is attempting to write a new value or invalidate the entry, update algorithms must synchronize all processors and do a cooperative TLB flush in order to guarantee consistency.

**B.I.5.1 TLB RAM Diagnostic Access Address Format**



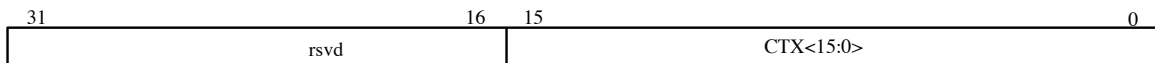
Field	Description
ENTRY	Entry in Page Descriptor Cache (1 of 64)
SEL	Selects portion of TLB to access. 0 = VA, 1 = Context, 2 = LOCK bit, 3 = PTE, 4 = Root Pointer (cached), 5 = level-2 PTP (cached), 6 = Level-2 Vaddr (cached)
rsvd	reserved: these bits are ignored

**B.I.5.1.2 VA Format (sel = 0 and sel = 6)**



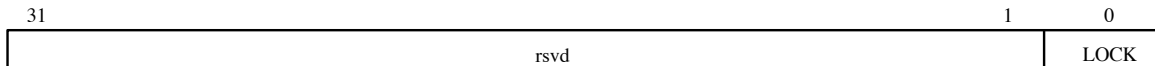
Field	Description	Type
VA<31:12>	Virtual address tag. Depending on PTE level, not all bits are significant.	RW
rsvd	reserved, read as '0'	R

**B.I.5.1.3 Context Format (sel = 1)**



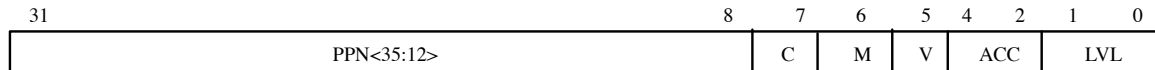
Field	Description	Type
CTX<15:0>	Context Tag.	RW
rsvd	reserved, read as '0'	R

**B.I.5.1.4 LOCK Format (sel = 2)**



Field	Description	Type
LOCK	If '1', the contents of this TLB entry will not be replaced	RW
rsvd	reserved, read as '0'	R

**B.I.5.1.5 PTE Format (sel = 3)**

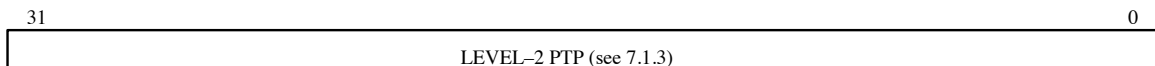


Field	Description	Type
PPN<35:12>	Physical Page Number	RW
C	Cacheable	RW
M	Modified	RW
V	Valid TLB entry	RW
ACC	Access Permission; smae as SRMMU	RW
LVL	PTE level; 0 = root, 1 = region, 2 = segment, 3 = page.	RW

**B.I.5.1.6 Root Pointer Cache Format (sel = 4)**



**B.I.5.1.7 Level-2 PTP Cache Format (sel = 5)**



**B.I.6 ASI's Implemented**

ASI	FUNCTION	SIZE
0x2	<i>Reserved in Viking/NE systems</i>	—
0x3	<b>Ref MMU Flush/Probe</b>	<b>Word</b>
0x4	<b>Module Registers</b>	<b>Word</b>
0x6	<b>SRMMU Diagnostic I/D-TLB</b>	<b>Word</b>
0x8	<b>User Instruction</b>	<b>All</b>
0x9	<b>Supervisor Instruction</b>	<b>All</b>
0xA	<b>User Data</b>	<b>All</b>
0xB	<b>Supervisor Data</b>	<b>All</b>
0xC	<b>I-\$ Cache Tag</b>	<b>Double</b>
0xD	<b>I-\$ Cache Data</b>	<b>Double</b>
0xE	<b>D-\$ Cache Tag</b>	<b>Double</b>
0xF	<b>D-\$ Cache Data</b>	<b>Double</b>
0x20-0x2F	<b>SRMMU bypass, PA&lt;35:32&gt; = ASI&lt;3:0&gt;</b>	<b>All</b>
0x30	<b>Store Buffer Tags</b>	<b>Double</b>
0x31	<b>Store Buffer Data</b>	<b>Double</b>
0x32	<b>Store Buffer Control</b>	<b>Single</b>
0x36	<b>I-cache Flash Clear</b>	<b>Word</b>
0x37	<b>D-cache Flash Clear</b>	<b>Word</b>
0x38	<b>MMU Breakpoint Diagnostics</b> <i>(See Viking Specification for details)</i>	<b>Double</b>
0x39	<b>BIST Diagnostics</b> <i>(See Viking Specification for details)</i>	<b>Word</b>
0x40-0x41	<b>Emulation temps [1-2]</b> <i>(See Viking Specification for details)</i>	<b>Word</b>
0x44	<b>Emulation Data In1</b> <i>(See Viking Specification for details)</i>	<b>Word</b>
0x46-4C	<b>Emulation Registers</b> <i>(See Viking Specification for details)</i>	<b>Word</b>

Alternate space accesses with reserved or unassigned ASI's will result in an error trap.

**B.I.7 Module Control Space Address Map**

PA<35:0>	Name
0xFFnFFFFFFC	<b>MBus Port Address Register for MID 'n'</b>

**B.I.8 IU PSR Number**

Bits <27:24> VER = 0x0  
Bits <31:28> IMPL = 0x4 (Texas Instruments)

**B.I.9 Module-Specific Quirks**

This module violates the MBus specification for coherent snooping with MIH\* in cycle A+2; instead it invokes MBus Specification Appendix B.7, and provides MIH\* in cycle A+3.

Since this is a module with physically addressed caches there is no VA superset provided in the address phase of MBus transactions.

**The Viking/NE module does not provide a place where the software can read the MID directly.** See 5.4.3 for a description of mechanisms for determining MID.

Unlike the Viking/E\$ module this module has no hardware support for Bcopy/Bfill.

If Viking/NE receives an error acknowledge from the MBus on a data store, it will post a level-15 module\_error broadcast interrupt, and will *also* take a store\_buffer\_exception (priority 2, trap type 0x2b).

Note that locking all TLB entries can lead to a deadlock (infinite table walk) and so must be avoided.

**B.II: Module Notes: Ross 605/64K**

*Based on Cypress Semiconductor 'SPARC RISC Users Guide' Second Edition, February 1990, with corrections based on conversations with Ross Semiconductor. Note that differences between this appendix and the Cypress book represent errata in the Cypress book!!*

- B.II.1 Module Overview
- B.II.2 Cache Details
- B.II.3 Module Registers: differences from core (section 4)
- B.II.4 Additional Registers Specific to this Module
- B.II.5 MMU Details
- B.II.6 ASI's Implemented
- B.II.7 Module Control Space Address Map
- B.II.8 IU PSR Number
- B.II.9 Module-Specific Quirks
- B.II.10 Module Write Buffers
- B.II.11 Exiting Boot State



**B.II.1 Module Overview**

The Ross 605 module consists of a chipset developed by Cypress Semiconductor/Ross Technologies. The processor set is a 1-scalar SPARC processor and cache system. The chipset includes the CY7C601 IU, a CY7C602 FPU, a CY7C605 CMU (Cache Controller and MMU) and two CY7C157 16KB x 16 pipelined cache SRAM. The entire chipset runs synchronous with the 40 MHz MBus clock.

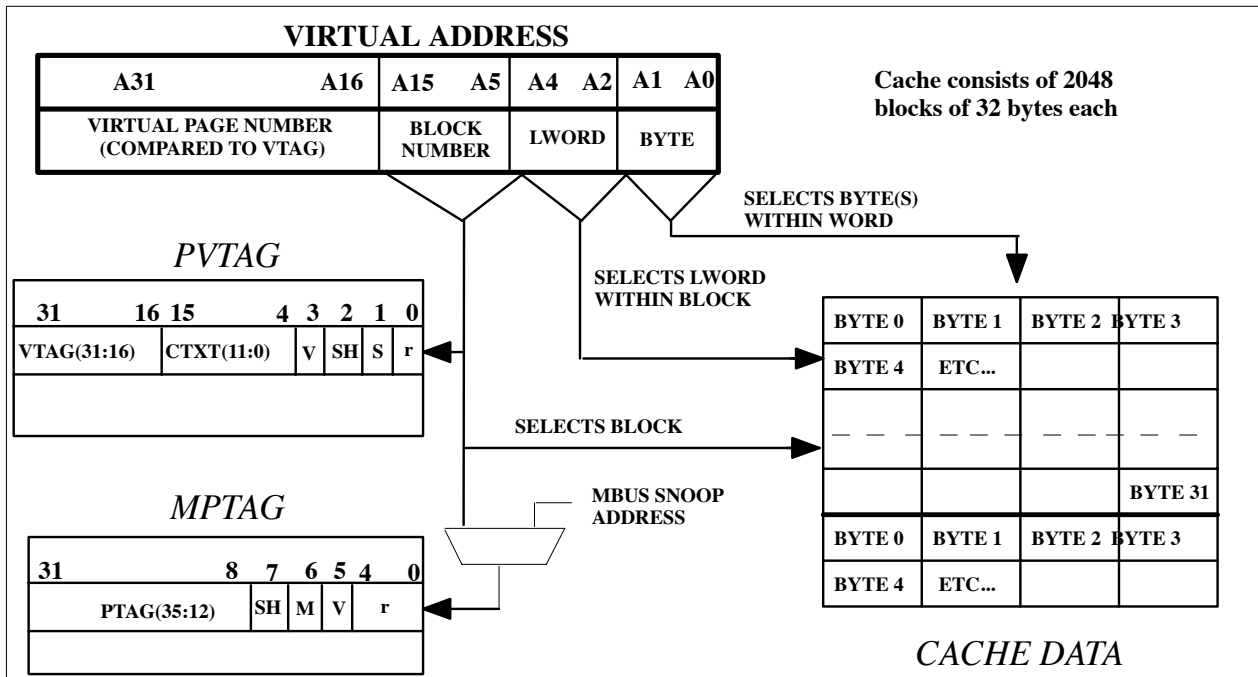
This logical module appears in two physical implementations, the Ross 6001 module (uniprocessor) and the Ross 6002 module (dual processor, i.e. two logical modules in one physical module). The 6001 is used for bringup purposes only, and the 6002 is used as a real product at Sun.

**B.II.2 Cache Details**

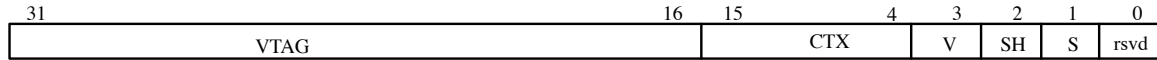
The Ross 605 system utilizes a 64 KB virtual-address direct-mapped cache, with one cache shared by both instruction access and data access. The cache participates in the MBus level-2 coherence protocols. Snooping is implemented via a virtually-indexed, physically tagged 'dual' directory. The data cache can be programmed to be either a write-through cache with a no-write-allocate policy or a write-back cache with write-allocate. The latter is recommended for use in Sun-4M systems.

The cache consists of 2048 blocks of 32 bytes each. The tag for each block in the main directory contains the *virtual tag* VA<31:16>, the *context number*, a *valid* bit and a *Supervisor* bit. The 'dual' directory contains the physical tag for that block, along with a *valid*, *shared*, and *modified* bit.

In write-back mode, aliases are detected by checking the physical address of the miss with the 'dual' tag on the current occupant of the cache block. If an alias is detected then the miss can be serviced without accessing the system memory. Cache alias size is 64 KB.

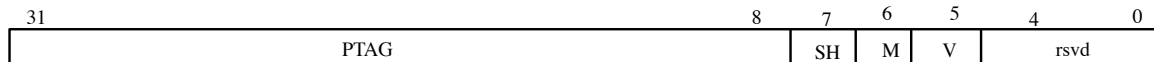


**B.II.2.1 Cache Virtual Tag Format (PVTAG)**



Field	Description	Type
VTAG	Virtual Tag VA<31:18>	RW
CTX	Context Number CTX<11:0>	RW
V	Valid cache entry	RW
SH	Shared	RW
S	Supervisor mode	RW
rsvd	reserved: read as '0'	R

**B.II.2.2 Dual Directory Tag Format (MPTAG)**



Field	Description	Type
PT	Physical Tag PA<35:12>	RW
SH	Shared block	RW
M	Modified	RW
V	Valid cache entry	RW
rsvd	reserved: read as '0'	R

**B.II.2.3 Addresses for Cache Diagnostic Access**

Cache Line	MPTAG (ASI = 0xE)	PVTAG (ASI = 0xE)	Data (ASI = 0xF)
0	0x00040000	0x00000000	0x00000000
1	0x00040020	0x00000020	0x00000020
2	0x00040040	0x00000040	0x00000040
3	0x00040060	0x00000060	0x00000060
...			
2047	0x0004FFE0	0x0000FFE0	0x0000FFE0

**B.II.2.4 Cache Flushing**

The cache in the Ross 605 module is flushed locally by each processor. Flushing involves use of the cache flush ASI's with *sta* accesses.

**B.II.2.5 Cache Snooping**

The Ross 605 provides VA<19:16> and monitors VA<15:12> in the MBus VA Suerprset field; this is sufficient to snoop a 64 KB cache. **An important note is that there is no way to disable snooping, even when the cache is disabled; this means that all cache tags in all modules must be invalidated prior to enabling any of the caches in the system.**

**B.II.3 Module Registers: differences from core (section 4)**

**B.II.3.1 Module Control Register: differences from core (section 4.1)**

Bit <10>: CB, rw. Only copy-back mode is used in Sun-4M, so this bit should be set to '1'.  
 Bit <11>: MR, rw. Memory Reflection: should always be set to '0' in Sun-4M usage.  
 Bits <18:15>: MID<3:0>, rw. Ross 605 does not read the pins of the module connector to establish the MID, so the MID must be set by boot firmware. This must be done very early in the boot process (see B.II.10).  
 Bits <23:19>, ro. Unassigned, read as '0'.  
 Bits <27:24>: SRMMU VER = 0xF, (CY7C605), ro  
 Bits <31:28>: SRMMU IMPL = 0x1 (Cypress Semiconductor), ro.

**B.II.3.2 Context Table Pointer Register: differences from core (section 4.2)**

The Context Table Pointer Register has bits <31:10> = CTP <35:14>, and bits <9:0> are reserved. This means that the context table must be 16KB-aligned in memory.

**B.II.3.3 Context Register: differences from core (section 4.3)**

N = 11, thus 4096 contexts are supported.

**B.II.3.4 Synchronous Fault Status Register: differences from core (section 4.4)**

The SFSR is clear-on-read in this implementation.

**B.II.3.5 Asynchronous Fault Status Register: differences from core (section 4.5)**

Clearing of the AFSR is controlled by reads of the AFAR. The AFSR should be read prior to the AFAR, and the AFAR should only be read if AFV (AFSR bit <0>) is asserted. This avoids a race condition between asynchronous faults being posted and accesses to the asynchronous fault registers.

**B.II.3.6 Reset Register: differences from core (section 4.6)**

This register is clear-on-read.

Snooping is maintained during either an SI or a WD reset.

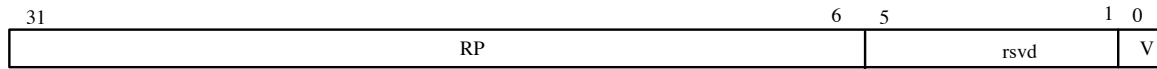
**B.II.3.7 MBus Port Address Register Register: differences from core (section 4.6)**

The CY7C605 does not support an MBus Port Register.

**B.II.4 Additional Registers Specific to this Module**

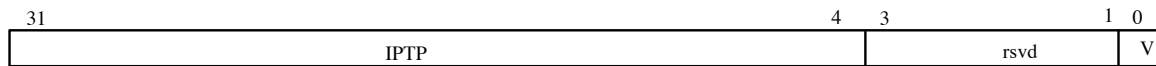
Address (ASI = 0x4)	Name
0x00001000	Root Pointer Register (RPR)
0x00001100	Instruction Access PTP (IPTP)
0x00001200	Data Access PTP (DPTP)
0x00001300	Index Tag Register (ITR)
0x00001400	TLB Replacement Control Register (TRCR)

**B.II.4.1 Root Pointer Register**



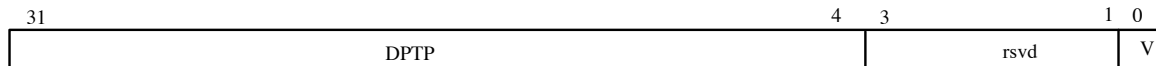
Field	Description	Type
RP	<b>Root Pointer: This is the Context level table PTP (page table pointer). It is part of the page table pointer cache. Valid. Cleared on power-on reset, or on writes to the Context Register or to the Context Table Pointer Register. Reserved.</b>	RW
V		RW
rsvd		R

**B.II.4.2 Instruction Access PTP Register**



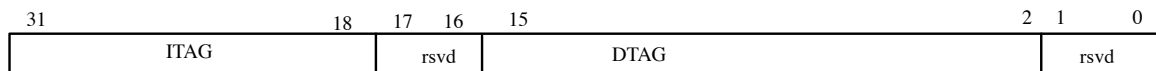
Field	Description	Type
IPTP	<b>Instnction Access PTP (page table pointer). Contains the level-2 PTP for IFETCH. It is part of the page table pointer cache. Valid. Cleared on power-on reset. reserved</b>	RW
V		RW
rsvd		R

**B.II.4.3 Data Access PTP Register**



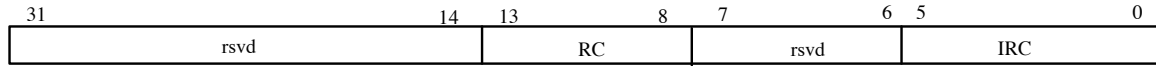
Field	Description	Type
DPTP	<b>Data Access PTP (page table pointer). Contains the level-2 PTP for Data access. It is part of the page table pointer cache. Valid. Cleared on power-on reset. reserved</b>	RW
V		RW
rsvd		R

**B.II.4.4 Index Tag Register**



Field	Description	Type
ITAG	<b>Tag for the IPTP register (level 1 and level 2)</b>	RW
DTAG		RW
rsvd		R

**B.II.4.5 TLB Replacement Control Register**



Field	Description	Type
RC	Replacement Counter for TLB random replacement	RW
IRC	Initial Replacement Counter	RW
rsvd	reserved	R

Both RC and IRC are reset to '0' upon power-on reset. In order to support TLB locking, the IRC can be set to a non-zero value. The IRC is used as an initialization value for RC; whenever RC reaches maximum count, it is pre-loaded with the value in IRC on the next increment. Locked TLB entries can be read/written through control space accesses (ASI = 0x6). When writing to the IRC field, write the same value to the RC field to ensure that the next-replacement pointer points to the unlocked area.

**B.II.5 MMU Details**

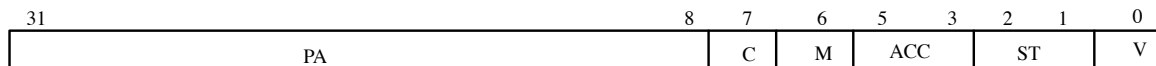
The MMU contains 64 TLB entries with a random replacement algorithm. The page tables are not cacheable and must be kept valid in main memory.

Upon the first miss after a context switch, an extra level of table-walk is supported in order to fetch that context's root pointer. A small cache keeps the most recently accessed PTP for the root level and one each for the level-2 PTP used for data and instruction access. This cache is visible as the RPR, IPTP, DPTP, and ITR registers. The IPTP and DPTP cache is flushed upon any TLB flush or upon any table walk for an instruction or data access, respectively. The entire PTP cache is flushed when the Context Register or the Context Table Pointer Register is written. A table walk is atomic, that is, MBSY\* is held asserted for the duration of a table-walk (up to 5 accesses to main memory in the case of a first-write to a page), and the LOCK bit in MBus address phase is asserted for table-walks.

PROBE\_ENTIRE is the only probe type supported by this SRMMU implementation.

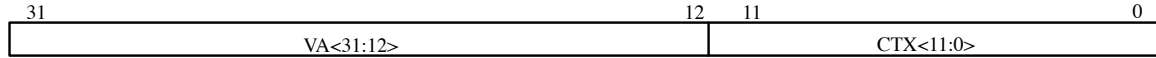
The IPTP and DPTP are not updated during table walks caused by address alias detection or copy-back flushes.

**B.II.5.1 TLB RAM Diagnostic Access Format**



Field	Description	Type
V	Valid Entry	RW
ST	Short translation Bits. 0 = page level and 3 = root.	RW
ACC	Access Permission (per SRMMU)	RW
M	Modified; this page has been written to	RW
C	Cacheable page	RW
PA	Physical address <35:12>; some or all is valid depending on the ST field.	RW

**B.II.5.2 TLB CAM Diagnostic Access Format**



Field	Description	Type
VA	Virtual Address tag for this entry	RW
CTX	Context tag for this entry	RW

**B.II.5.3 Addresses for TLB Diagnostic Access (ASI = 0x6)**

TLB Entry	TLB RAM	TLB CAM
0	0x00000000	0x00000004
1	0x00000008	0x0000000C
2	0x00000010	0x00000014
3	0x00000018	0x0000001C
...		
63	0x000001F8	0x000001FC

**B.II.6 ASI's Implemented**

Alternate space accesses with unsupported ASI's will be ignored (writes are ignored, reads provide garbage data).

This module has no hardware support for block copy or block fill.

Access with ASI = 0x1 is identical to access with ASI = 0x20 (Bypass, PA<35:32> = 0x0) with the exception that the MBL (local/boot mode) bit is asserted in the address phase. This bit is ignored in Sun-4M systems.

See the Ross 605 specification for details related to cache diagnostics, TLB diagnostics, TLB probes, cache flushing, and TLB flushing.

ASI	FUNCTION
0x1	Local Bus Mode (not used in Sun-4M)
0x3	Ref MMU Flush/Probe I/D-TLB
0x4	Module Registers
0x6	SRMMU Diagnostic I/D-TLB
0x8	User Instruction
0x9	Supervisor Instruction
0xA	User Data
0xB	Supervisor Data
0xE	I/D-\$ Cache Tag (A<18> = 0), MPTAG (A<18> = 1)
0xF	I/D-\$ Cache Data
0x10	Flush I/D cache by page
0x11	Flush I/D cache by segment
0x12	Flush I/D cache by region
0x13	Flush I/D cache by context
0x14	Flush I/D cache by 'user'
0x20-0x2F	SRMMU bypass, PA<35:32> = ASI<3:0>

### B.II.7 Module Control Space Address Map

The CY7C605 has no MBus slave port.

### B.II.8 IU PSR Number

Bits <27:24> VER = 0x1 (Cypress Semiconductor)  
Bits <31:28> IMPL = 0x1

### B.II.9 Module-Specific Quirks

The CY7C605 does not sense the MBus Module ID from the connector pins. Instead the MID must be initialized in software. The MID can be read in the MID Register; this register is provided specifically for this module.

The CY7C605 does not provide User/Supervisor information in the address phase of MBus transactions; for this reason any error status captured in system asynchronous error registers will *always* appear to be a *supervisor-mode* error, independent of the actual IU state.

CY7C605 support for reflective memory and for second-level cacheing is not used in Sun-4M systems.

Due to the nature of module write buffers and the way that asynchronous errors are captured the AFAR should not be read unless the AFV bit in the AFSR is asserted (see B.II.4.5).

### B.II.10 Module Write Buffers

The CY7C605 contains 32-bytes of data write buffer. When the chip is programmed to be a copy-back cache, the write buffer is used to hold 1 cache write-back (32 bytes, one address) or 1-4 non-cacheable writes (up to 4 double-word stores and up to four addresses). The write buffer maintains strong order among writes issued, and reads or shared cacheable writes (coherent invalidates) will stall the IU if the write buffers are not idle.

A read of a non-cacheable entity (such as a hardware register, i.e. the M-to-S AFSR) will guarantee draining of such write buffers at context switch.

### B.II.11 Exiting Boot State

#### B.II.11.1 Exiting Boot State With MMU On

The transition from boot mode to normal operation is a delicate time. In order to keep pipelines simple it is necessary to make the transition from boot-mode ifetch to instruction fetch translation (see 3.1.2) while executing from the same page; that is, the CY7C605 must be managed in a way that the same physical addresses are issued before, during, and after the change in the BM bit.

The physical address range for the EPROM is 0xFF000000 – 0xFF007FFF. Address bits <23:19> are don't-care (they are not decoded by the EPROM interface) and so they can have any value. In boot-mode Ifetch pass-thru mode, the VA-to-PA translation is PA<35:28> = 0xFF, PA<27:0> = VA<27:0>. This means that VA<31:28> are also don't-care when the module is in boot-mode. When boot mode is turned off, instruction fetches will be translated; the translation of this virtual address to the EPROM physical address must be established so that the fetches just before and just after the instruction that clears boot mode all fall on the same physical page. **Note that this means that PA<27:24> = VA<27:24> = 0x0, PA<18:0> = VA<18:0>, and the rest of the address can be freely selected.**

**B.II.11.2 Exiting Boot State With MMU Off**

It is also legal to exit boot-mode with the MMU disabled; when boot-mode is turned off and the MMU is off, all accesses happen in MMU pass-thru mode (see 3.1.2) where  $PA\langle 35:32 \rangle = 0x0$ , and  $PA\langle 31:0 \rangle = VA\langle 31:0 \rangle$ . In order to guarantee correct operation, the page in which boot mode is turned off must be copied from the EPROM to main memory at an address where  $PA\langle 18:0 \rangle$  is identical to  $PA\langle 18:0 \rangle$  in the EPROM address; the transition from boot mode to pass-thru will also involve a transition from EPROM fetch to main memory fetch.



## **B.III: Module Notes: Viking/E\$**

*Based on the Viking Microprocessor User Documentation, rev 2.00 11/01/90 and the Viking Cache Controller (MXCC) Specification (draft, 2/19/91). See those documents for details.*

- B.III.1      Module Overview
- B.III.2      Cache Details
  - B.III.2.1            Instruction Cache Details
  - B.III.2.2            Data Cache Details
  - B.III.2.3            External Cache Details
- B.III.3      Cache Coherence, Store Buffers, and Memory Models
  - B.III.3.1            Cache Coherence
  - B.III.3.2            Store Buffers
  - B.III.3.3            Memory Models
- B.III.4      Module Registers: differences from core (section 4)
- B.III.5      MXCC Registers (ASI = 0x2)
- B.II.6       MMU Details
- B.III.7      ASI's Implemented
- B.III.8      Module Control Space Address Map
- B.III.9      IU PSR Number
- B.III.10     Module-Specific Quirks

**B.III.1 Module Overview**

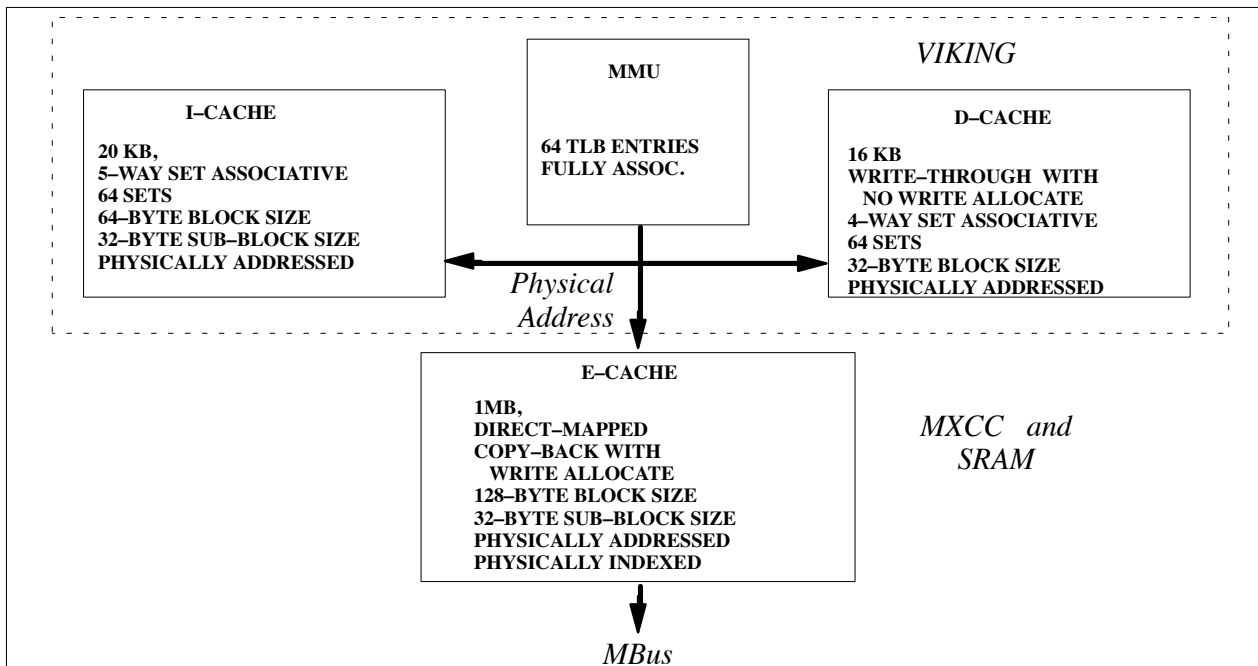
The Viking/E\$ module consists of a chipset developed jointly by Texas Instruments and Sun. The processor set is a superscalar SPARC processor and cache system. The chipset includes the Viking chip (TMS390Z50) with internal IU, floating-point, I- and D-caches, and MMU, the MXCC External Cache Controller (TMS390Z55), and 8 pipelined 128K x 9 cache SRAM (some versions may use 128K x 8). The MBus interface of the MXCC runs synchronous with the 40 MHz MBus clock, and the rest of the module runs synchronously at 50 MHz, with synchronization internal to the MXCC.

A physical Viking/E\$ module could have one or two Viking/E\$ logical modules on it. Both uni- and dual-processor modules will be built. (*Official names TBD*).

A similar module without the External Cache consists of a Viking chip connected directly to the MBus. That module is called Viking/NE, and it is defined in appendix B.I. The module type is identified by examining bit 11 of the Module Control Register (MCR). If the bit is '0' then this is the correct model; if the bit is a '1' then the Viking/NE appendix applies.

**B.III.2 Cache Details**

The Viking/E\$ system utilizes a Harvard architecture with a 20KB physical-address set-associative cache for instruction access, a 16KB physical-address set-associative cache for data access, and a 1MB physical-address direct-mapped second-level cache shared by both instruction access and data access. The caches participate in the MBus level-2 coherence protocols. Snooping is implemented via the second-level 'External' cache (E-\$) directory. The first-level data cache can be programmed to be either a write-through cache with a no-write-allocate policy or a copy-back cache with write-allocate. The former is required for Viking/E\$.



**B.III.2.1 Instruction Cache Details**

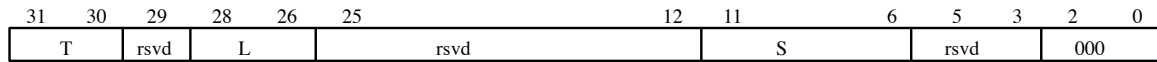
The I-cache is 5-way set associative. It consists of 64 sets with a block-size of 64 bytes, and each block is divided into two sub-blocks of 32 bytes each. Each set has an *Stag* (Set Tag) associated with it, and each line within the set has a *Ptag* (Physical Tag) associated with it.

The *Stag* contains a lock bit for each line within the set to allow selective locking of cache lines, with the exception of line 0x0; writes to that bit are ignored. Also visible in the *Stag* are the MRU (most-recently used) limited history bits used to implement the set replacement algorithm. Details of the algorithm are available in the Viking User's Guide. All bits can be read and written for diagnostic and locking purposes.

The Ptag for each block in the I-\$ directory contains the *physical tag* PA<35:12> for the block plus a *valid* bit for each sub-block. The I-\$ snoops write activity and will invalidate accordingly. If code modifies instructions, a SPARC *flush* must be executed to ensure consistency; the *flush* will synchronize on the completion of all pending writes, and will also flush the instruction prefetch buffer. The I-\$ can be initialized with a *flash-clear* mechanism.

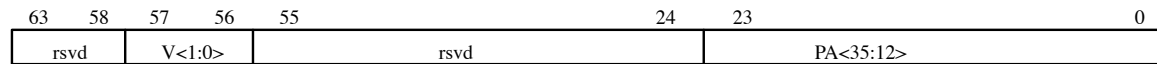
The state of I-cache tags and data are unaffected by Watchdog or system reset. The I-cache is enabled by the IE bit in the MCR.

**B.III.2.1.1 Instruction Cache Tag Address Format (ASI = 0xC)**



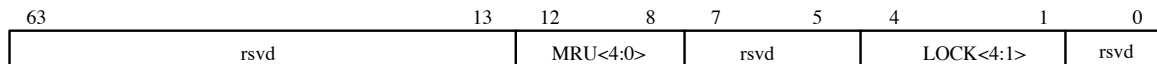
Field	Description
T	Type of tag; 1 = Set Tag (stag), 2 = Physical Tag (ptag), 0 = 3 = reserved
L	Line within set (0-4); (5-7) = reserved
S	Set (1 of 64)
rsvd	reserved: these bits are ignored

**B.III.2.1.2 Instruction Cache Ptag Format (64-bit access only)**



Field	Description	Type
V<1:0>	Valid bit for the 32-byte sub-blocks; V<1> corresponds to the high-order sub-block (A<4> = 1) and V<0> to the low-order.	RW
PA<35:12>	Physical tag address bits	RW
rsvd	reserved: read as '0'	R

**B.III.2.1.3 Instruction Cache Stag Format (64-bit access only)**



Field	Description	Type
MRU<4:0>	Access history bits used in the partial-LRU algorithm	RW
LOCK<4:1>	Lock bits for blocks <4:1> within the set.	RW
rsvd	reserved: read as '0'	R

**B.III.2.1.4 Instruction Cache Data Address Format (ASI = 0xD)**

31	29	28	26	25	12	11	6	5	3	2	0
rsvd	L		rsvd			S	DW		000		

Field	Description
L	<b>Line within set (0-4); (5-7) = reserved</b> <b>Set (1 of 64)</b> <b>Double-word within line</b> <b>reserved: these bits are ignored</b>
S	
DW	
rsvd	

**B.III.2.1.5 Instruction Cache Flash Clear (ASI = 0x36)**

31	30	0
T	rsvd	

Field	Description	Type
T	<b>Type: 0 = clear all Valid and MRU bits in PTags and Stags.</b> <b>1 = clear all Lock bits in Stags.</b> <b>reserved: ignored</b>	W
rsvd		W

**B.III.2.2 Data Cache Details**

The D-cache is 4-way set associative. It consists of 128 sets with a block-size of 32 bytes; there is no sub-blocking. Each set has an *Stag* (Set Tag) associated with it, and each line within the set has a *Ptag* (Physical Tag) associated with it. In the Viking/E-\$ module the D-\$ is configured as a write-through cache with a no-write-allocate policy.

The D-\$ in this configuration is the first-level cache in a 2-level cache hierarchy. The E-\$ tags maintain full inclusion on the contents of the D-\$ and I-\$, and so the E-\$ is able to snoop the MBus on behalf of the first-level caches. In the case of a snoop hit the MXCC will send a *coherent invalidate* transaction on the Viking bus, which will be snooped by the D-\$ if the SE bit in the MCR is set, and the D-\$ will invalidate the line as appropriate.

The *Stag* contains a lock bit for each line within the set to allow selective locking of cache lines, with the exception of line 0x0. Also visible in the *Stag* are the MRU (most-recently used) limited history bits used to implement the set replacement algorithm. Details of the algorithm are available in the Viking User's Guide. All bits can be read and written for diagnostic and locking purposes.

The *Ptag* for each block in the D-\$ directory contains the *physical tag* PA<35:12> plus a *valid*, *dirty*, and *shared* bit for the block. In Viking/E\$ the *shared* and *dirty* bits are not used. The D-\$ can be initialized with a *flash-clear* mechanism. The state of D-cache tags and data are unaffected by Watchdog or system reset.

**B.III.2.2.1 Data Cache Tag Address Format (ASI = 0xE)**

31	30	29	28	27	26	25	12	11	5	4	3	2	0
T	rsvd		L	rsvd			S		rsvd		000		

Field	Description
T	<b>Type of tag; 1 = Set Tag (stag), 2 = Physical Tag (ptag), 0 = 3 = reserved</b> <b>Line within set (0-3)</b> <b>Set (1 of 128)</b> <b>reserved: these bits are ignored</b>
L	
S	
rsvd	



spatially local stores into fewer, larger stores) the prefetch feature is expected to accelerate certain classes of numerical algorithms that tend to access large sequential arrays. More details can be found in the Viking User's Guide.

**B.III.2.3 External Cache Details**

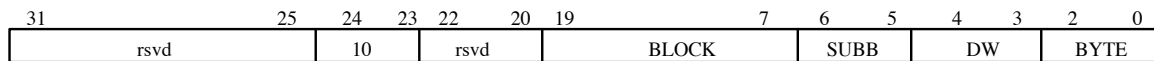
The E-\$ is a direct-mapped cache consisting of 8K blocks of 128 bytes each; each block is divided into 4 sub-blocks of 32 bytes. The E-\$ operates as a copy-back cache with a write-allocate policy. E-\$ tags contain the physical address bits PA<35:19> for the block, plus a *shared*, *owned*, *valid*, and *pending* bit for each of the four sub-blocks. The E-\$ is enabled by the CE bit in the MXCC Control Register.

The E-\$ maintains level-2 MBus consistency by snooping on physical addresses. The E-\$ directory is time-shared between Viking and MBus snoop access; there is no dual directory in this module. Coherent activity initiated on the MBus by the E-\$ does not provide meaningful VA Superset bits in the MBus address.

Diagnostic and control access to the E-\$ are made in space ASI = 0x2.

Address (ASI = 0x2)	Name
0x01000000 – 0x010FFFFFFF	E-cache data
0x01800000 – 0x018FFFFFFF	E-\$ tag
0x01C00000 – 0x01CFFFFFFF	MXCC Control Registers

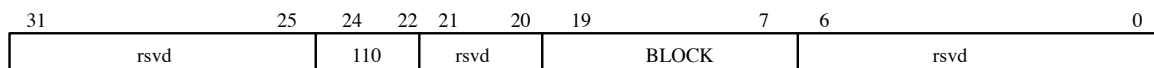
**B.III.2.3.1 E-Cache Data Address Format (ASI = 0x2)**



Field	Description
<b>BLOCK</b>	Cache block number
<b>SUBB</b>	Sub-block number
<b>DW</b>	Double-word within block
<b>BYTE</b>	byte within double-word
rsvd	reserved: these bits are ignored

E-\$ data can be accessed as byte, half-word, word, or double-word.

**B.III.2.3.2 E-Cache Tag Address Format (ASI = 0x2)**



Field	Description
<b>BLOCK</b>	Cache block number
rsvd	reserved: these bits are ignored

E-\$ tags can be accessed as double-words only.

**B.III.2.3.3 E-Cache Tag Format**

63	36	35	19	18	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
rsvd	PA<35:19>		rsvd	S3	O3	V3	P3	S2	O2	V2	P2	S1	O1	V1	P1	S0	O0	V0	P0		

Field	Description	Type
PA<35:19>	Physical tag for E-\$ line	RW
S<3:0>	Shared bit, sub-blocks <3:0>	RW
O<3:0>	Owned bit, sub-blocks <3:0>	RW
V<3:0>	Valid bit, sub-blocks <3:0>	RW
P<3:0>	Pending bit, sub-blocks <3:0>: Viking operation is pending	RW
rsvd	reserved: read as '0', writing has no effect	R

**B.III.2.3.4 E-Cache Data Prefetching**

E-\$ supports sequential data prefetch. This feature is enabled when the PF bit in the MXCC Control Register is set to 1. Prefetch will occur when MXCC detects a burst read access to data whose next sequential sub-block is not valid. Prefetch is bounded by block boundaries. Only one prefetch can be pending at any time.

**B.III.2.3.5 E-Cache Parity Protection**

The E-\$ has optional byte parity protection. A module that has parity will have 128K x 9 SRAM instead of 128K x 8. In order to determine if this is the case the following algorithm must be used, executing out of I-\$ with no intervening Viking bus cycles:

- 1) Enable parity checking in Viking by setting MCR bit <12> to '1' and in MXCC by setting MXCC Control register bit <3> to '1'.
- 2) Write a double-word to the E-\$ using diagnostic access; do a *stda* to ASI 0x2 with address 0x01000000 and data of 0xaaaaaaaaaaaaaa.
- 3) Write a double-word to an MXCC tag with opposite parity; do a *stda* to ASI 0x2 with address 0x01800000 and with data of 0x5151515151515151. The holding amplifiers on the bus will hold this parity.
- 4) Read back the data from the E-\$ by issuing a *ldda* at address 0x01000000. If the SRAM has parity bits then the data will come back correctly; if not then the IU will take a *data\_access\_exception* trap with the cause being a parity error, as indicated in the SFSR bit <14>.

If the module does support parity on the E-\$ then the SRAM must be initialized by writing to each double-word once in diagnostic mode using a *stda* at ASI 0x2 to the correct addresses (see B.III.2.3.1 for address information). All transactions between Viking and the MXCC and the E-\$, and between the MXCC and the E-\$ will be parity protected. If parity is not supported then the bits set in step (1) must be cleared.

**B.III.3 Cache Coherence, Store Buffers, and Memory Models**

**B.III.3.1 Cache Coherence**

**B.III.3.1.1 Cache Flushing**

Since Viking caches are physically addressed the contents of cache lines always corresponds to a backing store in physical memory. For this reason no flush mechanism is supported; there is never a need to dereference the contents of any cache line.

For diagnostic purposes, flushes can be forced by accessing a different physical address that maps to the same cache line, i.e. a displacement flush. An E-\$ line is flushed by reading a cacheable address that is identical modulo [1MB]. A D-\$ set can be flushed by reading in 4 cacheable addresses that map to the same set, i.e. modulo [4KB], or by using the D-\$ flash-clear mechanism. Similarly the entire I-\$ can be flushed with the flash-clear operation.

### B.III.3.1.2 Cache Aliasing

Since all caches in this system are physically addressed there is no aliasing rule needed.

### B.III.3.1.3 Cache Snooping

Viking/E\$ maintains consistency in both the first-level I-\$ and D-\$ and the second-level E-\$ by inclusion; the E-\$ always acts as backing store for the first-level caches, so the E-\$ is always able to snoop MBus coherent transactions on behalf of the internal caches. The only message necessary from the E-\$ to the I-\$ and D-\$ is a simple 'invalidate' with a physical address.

When the E-\$ is disabled the MXCC will forward all *coherent invalidate*, *coherent read invalidate*, and *coherent write invalidate* operations on the MBus as a *coherent invalidate* on the Viking bus. Note however that when the E-\$ is disabled all I-\$ and D-\$ misses will be sent to the MBus as a level-1 (non-coherent) transaction since the block sizes are different; **for this reason the E-\$ should be enabled before the I-\$ and D-\$.**

Also, the I-\$ always snoops stores from the D-\$ side, and maintains consistency. The SPARC *flush* instruction merely forces the IU to wait until all store queues have drained to ensure that the I-\$ is consistent.

### B.III.3.2 Store Buffers

#### B.III.3.2.1 Store Buffer Operation

Viking implements a store buffer which behaves as a FIFO; internally it is configured as a fully-associative cache of 8 double-words. Snoops from the IU use the fully-associative path. This store buffer eliminates most penalties for the write-through operation of the D-\$. All stores that have successfully been translated by the MMU are immediately placed into the store buffer, allowing the processor to continue with no store penalty. The IU will stall on a store only if the store buffer is full or disabled, or if a TLB miss occurs.

Stores that are *not* placed in the store buffer include atomic operations, store-alternate operations, updates of 'R' and 'M' bits in the MMU tables, and stores when the store buffer is disabled. Any of these stores will stall the IU, and will not occur until any pending stores in the store buffer have been drained to the E-; this guarantees that non-cached and bypass operations all behave in a strongly ordered manner.

While the store buffer is structured as a cache, it operates as a FIFO for store operations. Stores will be completed in the order issued. If the (doubleword) address of a load matches the (doubleword) address of a store that is currently in the buffer, the load will stall until that write has completed. If an instruction fetch address matches a store buffer entry in that way, the fetch must wait until the store buffer has drained that entry to the E-; data is not forwarded from the store buffer to the instruction pipeline.

The store buffer also does burst collection. While it does not accumulate bytes, it will turn sequences of memory references within a cache block into burst write operations to the Viking bus. This burst can be of arbitrary size within a cache line. Order of issue is maintained, and the burst will stop at any access that is to a different cache line, so TSO can be maintained. There is no way to use the store buffer without also using burst collection.



The store buffer is automatically drained upon a context switch. If a store buffer exception occurs as a result of the copyout, the trap will be reported to the IU *before* the *sta* to the context register completes, so the contents of the CTX register will reflect the context that owns the store buffer trap. Upon resuming execution after the trap, the store to CTX will complete.

The store buffer is enabled by the SB bit in the MCR.

**B.III.3.2.2 Store Buffer Exceptions**

When a fault occurs on a buffered store, it is reported to the IU as a *data\_store\_error* trap (priority 2, type 0x2b). In order for the error to be reported the NF bit in the MCR must be 0x0 and traps must be enabled. Upon the detection of a store buffer fault (due to a report of an error from the E-\$ or the MBus) the store buffer is frozen, the SB bit in the MCR is cleared, and the store buffer will retain all pending stores including the faulting one. A buffer copyout is *not* initiated. All subsequent store accesses are synchronous and behave as if the store buffer is disabled. Nominally these stores belong to the trap handler, and so belong to a different, unrelated thread of execution. Note that this is a synchronous trap issued for an asynchronous event, similar to an interrupt.

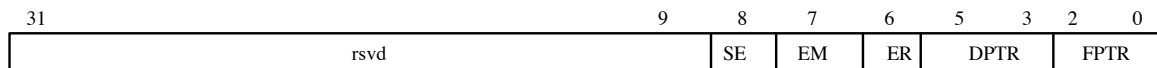
The trap handler can retry the store operation by obtaining the data, address, size, and cacheability information from the store buffer using diagnostic access. The store can be recreated by using an ASI bypass with *sta*; the AC bit in the MCR should be set to match the C bit in the store buffer tag. The trap handler should set the NF bit prior to retrying the store. After the *sta* the handler can examine the SFSR to determine the error status. After the fault status is determined the OS can decide what action to take.

Alternatively the trap handler can re-issue the faulted store by simply re-enabling the store buffer. When a store buffer exception occurs, the store buffer pointers retain their present value. Setting NF=0 will prevent another store buffer trap from being taken if the store sees another exception. In this case, the store buffer will still turn off, even with NF=0. However, the store buffer error remains pending, and the IU-pipe doesn't see it until NF=1. A bit in the store buffer control register indicates the presence of a pending (unacknowledged) store buffer error, or the code could check if the store buffer is still enabled after the store.

Store order can be maintained even in the presence of an error; the trap handler simply retries all pending stores in the order they were placed in the buffer.

After handling an error, the trap handler can return the store buffer to operation by clearing all valid bits, setting the *drain* pointer equal to the *fill* pointer in the Store Buffer Control Register, and then setting the SB bit in the MCR to 0x1.

**B.III.3.2.3 Store Buffer Control Register (ASI = 0x32)**

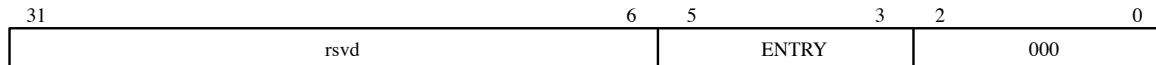


Field	Description	TYPE
SE	<b>Store Buffer Enabled: shadow of bit in MCR</b>	R
EM	<b>Store Buffer Empty: 1 = Empty. Set to '1' at reset.</b>	R
ER	<b>Store Error Pending: set when a store buffer error occurs when traps are disabled. Cleared when the trap is taken.</b>	R
DPTR	<b>Drain Pointer: indicates the buffer entry that is at the head of the queue</b>	RW
FPTR	<b>Fill Pointer: indicates the next entry that will be written to.</b>	RW

The store buffer is full if the fill pointer equals the drain pointer and there are valid entries. The buffer is empty if the two pointers are equal and there are no valid entries. The pointers increment modulo 8.

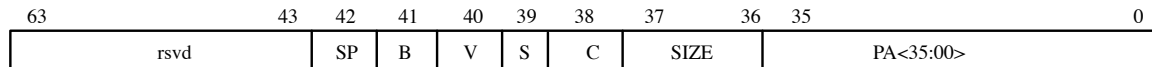
**B.III.3.2.4 Diagnostic Access to the Store Buffer**

**B.III.3.2.4.1 Store Buffer Tag Address Format (ASI = 0x30)**



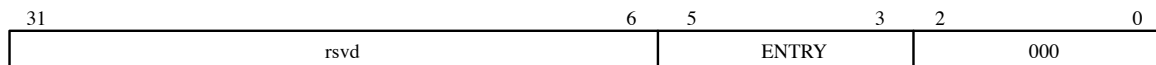
Field	Description
ENTRY rsvd	Store buffer entry number reserved: these bits are ignored

**B.III.3.2.4.2 Store Buffer Tag Format**



Field	Description
SP	Store Barrier Pointer: Used in PSO to mark the synchronization point
B	Burst Mode Access: Indicates that the next entry in the buffer corresponds to the next address, and thus can be issued in a burst
V	Valid entry
S	Supervisor: Indicates that the store was issued by a supervisor thread
C	Cacheable: Indicates that this store is a cacheable access
SIZE	Size of access: 00 = byte, 01 = half-word, 10 = word, 11 = double-word
PA<35:00>	Physical address of the store. Must be size-aligned or an error will occur
rsvd	reserved: these bits are ignored

**B.III.3.2.4.3 Store Buffer Data Address Format (ASI = 0x31)**



Field	Description
ENTRY rsvd	Store buffer entry number (double-word access) reserved: these bits are ignored

**B.III.3.3 Memory Models**

Viking is capable of operating in three modes of ordering: *strong sequential ordering*, *Total Store Ordering* (TSO), and *Partial Store Ordering* (PSO). The first is the standard model of data ordering in which all loads and stores complete in order of issue. The latter two models are those defined in the SPARC Architecture Manual version 8.

Strong ordering is achieved by disabling the internal store buffer. This is done by setting the SB bit in the MCR to 0x0. TSO is enabled by setting the SB bit to 0x1 and the PSO bit in the MCR to 0x0. PSO is enabled by setting both SB and PSO to 0x1. TSO is the nominal memory model in Sun-4M and in SPARC V.8. PSO requires explicit synchronization of stores; stores are guaranteed to be ordered only with respect to the SPARC synchronization instruction *stbar*.

**B.III.4 Module Registers: differences from core (section 4)****B.III.4.1 Module Control Register: differences from core (section 4.1)**

Bit <7>: PSO, rw. 1 = PSO mode, 0 = TSO mode. **Always set to '0' in Sun-4M systems.**

Bit <8>: DE, rw. Data-cache enable

Bit <9>: IE, rw. Instruction-cache enable.

Bit <10>: SB, rw. Store buffer enable.

Bit <11>: MB, ro. 0 = E-cache mode, this is a Viking/E\$ module and the D-\$ is write-through. 1 = MBus mode, this is a Viking/NE module and the D-\$ is copy-back mode (in that case appendix B.I applies).

Bit <12>: PE, rw. Parity check enable; turns on parity checking on Viking and cache RAM access. This should only be used if the E-\$ has parity bits (see B.III.2.3.5).

Bit <13>: BT, rw. Boot mode. 0 = normal operation, 1 = boot mode. This bit functions exactly like bit <14> in the generic register specification in section 4.1.

Bit <14>: SE, rw. Snoop Enable: when '1' snooping is enabled for the I-\$ and D-\$.

Bit <15>: AC, rw. Alternate cacheable. When the MMU is disabled or when the PTE is not used for translation (i.e. for MMU bypass accesses) this bit provides the cacheability status of all transactions **with the exception of boot-mode instruction fetches, which are never cacheable.** This bit affects both the internal I-\$ and D-\$ and the external E-\$. This functions differently from some modules, which will use the AC bit as an advisory for the MBus address phase but will not cache these references. In Sun-4M machines care must be taken to not accidentally attempt cacheable accesses to non-memory devices or an error acknowledge will result.

Bit <16>: TC, rw. Table-walk cacheable. When '1' table-walk references can be cached in the E-cache. Table-walk data is *never* cached in the internal cache.

Bit <18>: PF, rw. Data Prefetch enable (see section B.III.2.2.6)

Bits <27:24>: SRMMU VER = 0x0, ro. Mask revisions will show only in the IU PSR.

Bits <31:28>: SRMMU IMPL = 0x4, (Texas Instruments), ro

**B.III.4.2 Context Table Pointer Register: differences from core (section 4.2)**

The Context Table Pointer Register has bits CTPR <31:8> = CTP <35:12>, and bits CTPR <7:0> are reserved. Depending on the number of contexts supported the CTP may have any number of bits from <18:12> zeroed; the context table must be size aligned. CTP <17:12> is the logical **OR** of CTPR <13:8> and CTX <15:10>. Thus if there are 10 context bits, CTPR<31:8> represent CTP<35:12> and the context register bits CTX<9:0> represent CTP<11:2>, with CTP<1:0> = 00; the context table would have to be 4K aligned. At the far end of the spectrum, if there are 16 context bits then CTPR<31:14> represent CTP<35:18>, CTX<15:0> = CTP<17:2>, and again CTP<1:0> = 00; then the context table would have to be 256K aligned. Any size in between is also supported. At a minimum the table must be 4K aligned.

**B.III.4.3 Context Register: differences from core (section 4.3)**

N = 15, thus up to 64K contexts are supported.

**B.III.4.4 Synchronous Fault Status Register: differences from core (section 4.4.1)**

Bit <13>: VMP: Viking Master Parity Error. Set when the MXCC detects a parity error on data sent from Viking to MXCC on the Viking bus when Viking is the bus master. Indicates broken hardware.

Bit <14>: P: Parity Error. Indicates that a parity error has been detected on access to the E-\$ or in communication with the MXCC. If this bit is set, bit <12> (Uncorrectable Error) will also be set (see section B.III.2.3.5).

Bit <15>: SB: Store Buffer Error. Indicates that a store buffer error has been detected (see section B.III.3.2.2)

Bit <16>: CS: Control Space Access Error. Set if a *lda*, *sta*, or *swapa* returns an access error, except for bus error on ASI's 0x8 – 0xB and 0x20 – 0x2F or for bus error on MMU probe operations. Normally these errors will occur if there is an invalid ASI space, an incorrect size of access, an invalid address within a valid ASI space, or a bus error on accessing space ASI = 0x2. MFAR will hold the correct address in case of a CS error, although the ASI is not captured anywhere.

Bit <17>: EM: Error Mode Reset Taken. When set this indicates that the processor has taken a watchdog reset. In Viking/E\$ this status also can be found in the MXCC Reset Register bit <2>.

**B.III.4.5 Synchronous Fault Address Register: differences from core (section 4.4.2)**

As allowed in the SRMMU specification this register will never hold the address of an instruction fault; that information will be found in the saved PC/NPC. A special diagnostic path allows for this register to be written as well as read at ASI = 0x4, VA = 0x00001400.

**B.III.4.6 Asynchronous Fault Status and Address Registers: differences from core (section 4.5)**

Viking does not support an AFSR/AFAR, even though it contains write buffers which allows write operations to complete asynchronous to the instruction flow. Any error reported asynchronously will be captured in MXCC registers. See B.III.3.2.2 'Store Buffer Exceptions' for more information.

**B.III.4.6 Reset Register: differences from core (section 4.6)**

There is no Reset Register internal to Viking. The Reset register resides in the MXCC in ASI = 0x2 space.

Assertion of WD will cause assertion of the Module Error output. Snooping is maintained during either an SI or a WD reset.

**B.III.4.7 MBus Port Address Register Register: differences from core (section 4.6)**

The MBus Port Address Register is defined as part of the MXCC register set in ASI = 0x2 space.

**B.III.5 MXCC Registers (ASI = 0x2)**

**B.III.5.4 MXCC Registers**

Address (ASI = 0x2)	Name
0x01C00000	Stream Data
0x01C00100	Stream Source
0x01C00200	Stream Destination
0x01C00300	Reference/Miss Count
0x01C00400	Interrupt Pending ( <i>Not Used in MBus systems</i> )
0x01C00500	Interrupt Mask ( <i>Not Used in MBus systems</i> )
0x01C00600	Interrupt Pending Clear ( <i>Not Used in MBus systems</i> )
0x01C00700	Interrupt Generation ( <i>Not Used in MBus systems</i> )
0x01C00800	BIST (Built-in Selftest)
0x01C00900	Reserved
0x01C00A00	MXCC Control
0x01C00B00	MXCC Status
0x01C00C00	Module Reset
0x01C00D00	Reserved
0x01C00E00	Error Registers
0x01C00F00	MBus Port Address Register

Note: All MXCC registers are accessed as 8-byte (double-word) only.

**B.III.5.4.1 Block Copy and Block Fill Operations**

When configured for MBus use, the MXCC supports physically-addressed block copy or block fill operations on 32-byte blocks. *Block copy* operations are done by filling the stream data register via a *stream read*, then writing it to the destination with a *stream write*. A *stream read* is issued by writing the block address to the Stream Source Register; a *stream write* is issued by writing the destination block address to the Stream Destination Register. A *block fill* is performed by writing the fill data to the Stream Data Register, then issuing a series of *stream writes*. The IU can issue a continuous series of *stream writes* and *stream reads*; hardware will interlock on pending operations.

**B.III.5.4.1.1 Stream Source Register and Stream Destination Register Format**

63	62	37	36	35	5	4	0
RDY	rsvd			C	PA<35:5>		00000

Field	Description	TYPE
RDY	Ready: indicates that a pending stream read or stream write (Source or Destination Register, respectively) has completed. Writes ignored.	R
C	Cacheable: indicates if the data is to/from cacheable space.	RW
PA<35:5>	Physical address of the block to transfer.	RW
rsvd	reserved: these bits are ignored, read as '0'	R

**B.III.5.4.1.2 Stream Data Register Address Format**

31	6	5	3	2	0
rsvd			DW	rsvd	

Field	Description
DW	Selects the double-word within the 32-byte block to be written
rsvd	reserved: these bits are ignored, use '0'

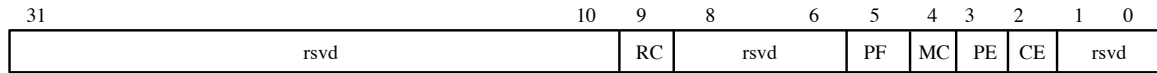
**B.III.5.4.2 Reference/Miss Count Register**

63	32	31	0
CMC		CRC	

Field	Description	Type
CMC	Cache Miss Count; increments on each E-\$ miss	RW
CRC	Cache Reference Count; increments on each E-\$ reference	RW

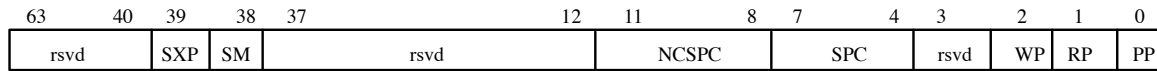
This register is used to track the E-\$ hit ratio. The counters are initialized by writing 0x0 to this register. When bit <31> becomes a '1' (in approximately 43 seconds) both CRC and CMC will freeze until bit <31> is cleared by software (normally the entire counter will be reset by storing 0x0 again).

**B.III.5.4.3 MXCC Control Register**



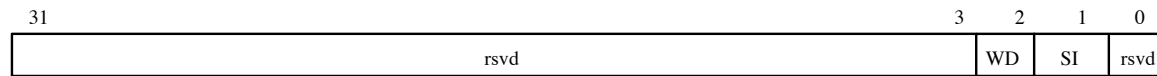
Field	Description	Type
<b>RC</b>	<b>Read Reference Count: when set, only read references are counted in the Reference/Miss Counter</b>	<b>RW</b>
<b>PF</b>	<b>Prefetch Enable; triggers prefetch on burst read miss if next subblock is not in the E-<math>\\$</math>; stops at block boundaries.</b>	<b>RW</b>
<b>MC</b>	<b>Multiple Command Enable: normally set to '1'. Diagnostic.</b>	<b>RW</b>
<b>PE</b>	<b>1 = generate and check even parity. 0 = generate odd parity, checking is disabled. '0' can be used to force parity errors.</b>	<b>RW</b>
<b>CE</b>	<b>Enable E-cache</b>	<b>RW</b>
<b>rsvd</b>	<b>reserved; writes are ignored, reads as '0'</b>	<b>R</b>

**B.III.5.4.4 MXCC Status Register**



Field	Description	Type
<b>SXP</b>	<b>Store Exception Pending (<i>definition is unclear..waiting for info</i>)</b>	<b>R</b>
<b>SM</b>	<b>Synchronous Mode: Shows if Viking is running at MBus speed.</b>	<b>R</b>
<b>NCSPC</b>	<b>Non-cacheable store pending count</b>	<b>R</b>
<b>SPC</b>	<b>Store-Pending Count</b>	<b>R</b>
<b>WP</b>	<b>Write Miss Pending</b>	<b>R</b>
<b>RP</b>	<b>Read Miss Pending</b>	<b>R</b>
<b>PP</b>	<b>Prefetch Pending</b>	<b>R</b>
<b>rsvd</b>	<b>reserved; writes are ignored, reads as '0'</b>	<b>R</b>

**B.III.5.4.5 Module Reset Register**



Field	Description	Type
<b>WD</b>	<b>Watchdog Reset; write '1' to clear WD and to deassert the Module-Error Interrupt.</b>	<b>R †</b>
<b>SI</b>	<b>Software Internal Reset</b>	<b>RW</b>
<b>rsvd</b>	<b>Reserved, reads a '0', writing has no effect.</b>	<b>R</b>

† Write '1' to clear.

Software internal reset is provided for diagnostic purposes, and is not used for normal system operation.

**B.III.5.4.6 MXCC Error Register**

63	62	61	60	59	58	57	56	47	46	39	38	37	36	35	0
ME	rsvd	CC	VP	CP	AE	EV	CCOP<9:0>	ERR<7:0>	S	rsvd	PA<35:0>				

Field	Description	Type
ME	If any of the 4 errors (CC,VP,CP,AE) occur and the bit is already set, the ME bit is set.	R
CC	Cache Consistency Error (unexpected E-\$ tag status)	R
VP	Viking Bus Parity error on a Viking write (Viking is master)	R
CP	Viking Bus Parity Error on MXCC read of E-\$ (from MBus)	RW
AE	Asynchronous Error: Error from MBus on write or stream operation which was already ack'ed to Viking.	R
EV	Error Valid: Indicates that this register contains error info.	R
CCOP<9:0>	CC Operation Code related to the error (see MXCC spec.)	R
ERR<7:0>	Error code in ERR<2:0> or cache parity syndrome DPAR<0:7>	R
S	1 = Supervisor, 0 = User mode IU when error occurred	
PA<35:0>	Physical address of the error	
rsvd	reserved; writes are ignored, reads as '0'	

Individual bits in this register are write-1-to-clear. This register is not affected by system reset. To initialize it, write to this register with data that is all 1's, that is 0xFFFFFFFFFFFFFFFF.

When CC, CP, or AE is set, the CCOP<7:0>, ERR<2:0>, S, and PA<35:0> fields are meaningful; the EV bit will be set and the module error interrupt will be asserted. In order to clear the error(s) that are posted, to deassert the Module\_error interrupt, and to release the error register so that new error status can be captured, write back to this register with the exact error information that was read; that way if a new error occurs between the read and the write to the error register the corresponding error bit will still be set, and the module\_error interrupt will still be asserted. Note however that the EV bit will not be set since those fields were frozen at the time the new error occurred, and status related to the new error is lost (the only information is CC, CP, or AE).

If the ME bit is set, or if more than one of CC, CP, and AE is set then multiple errors have occurred and some error status has been lost.

The CCOP<9:0> field indicates detailed status of the error that is posted. (At this time the decode of the CCOP field is not available; after production release, see the MXCC chip spec for further details).

If VP is set, the SFSR bit <13> will be set, and the SFAR will contain the corresponding address.

In the event of an AE, the ERR<7:0> field contains the error code in bits ERR<2:0>, defined as: 0x1 = Uncorrectable Error, 0x2 = Timeout Error, 0x3 = Bus Error, 0x4 = Undefined Acknowledge Error. These codes represent the acknowledges received from the MBus. Codes 0x0, 0x5, 0x6, and 0x7 are not meaningful and should never occur when EV is set. Bits ERR<7:3> are not defined for AE.

For a CP error the ERR<7:0> field represents the parity syndrome for the E-\$ access. ERR<7> reflects a parity error on data bits <63:56>, ERR<6> shows a parity error in data <55:48>, etc.

The decode of valid CCOP codes for MBus follow:

CCOP<9:0>				COMMAND	MBUS COMMAND	Note
Dcmd	rply	Pl	Xdst			
00110	1	1	010	Burst Read Miss Reply	CR	E\$ ON
00110	1	1	001	Read Miss Reply	CR	
00110	1	1	000	Write Miss Reply	CI	
00101	1	x	000	Shared Write Reply	CI or CRI †	
10101	1	x	001	Shared LDST Reply	CI or CRI [	
00010	1	1	000	Stream Read Reply	CR	
00111	1	0	000	Stream Write Reply	CWI	
00110	1	1	011	Prefetch Reply	CR	
01000	1	0	000	NC Read Reply	NCR	
01001	1	0	000	NC Write Reply	NCW	
01010	1	1	000	NC Stream Read Reply	NCR	
01011	1	0	000	NC Stream Write Reply	NCW	
11001	1	0	001	NC LDST Reply	NCR + NCW	
10011	1	0	000	Flush Reply ACK	NCW	
00110	1	1	100	Burst Read Reply	NCR	E\$ OFF
00110	1	1	101	Read Reply	NCR	
00101	1	0	010	Write Reply	NCW	
10101	1	0	011	LDST Reply	NCR + NCW	
00010	1	1	000	Stream Read Reply	CR	
00111	1	0	000	Stream Write Reply	CWI	
01000	1	0	000	NC Read Reply	NCR	
01001	1	0	000	NC Write Reply	NCW	
01010	1	1	000	NC Stream Read Reply	NCR	
01011	1	0	000	NC Stream Write Reply	NCW	
11001	1	0	001	NC LDST Reply	NCR+NCW	
10011	0	0	000	Flush ‡	CR or CRI	Initiated from Mbus side
00110	0	0	011	Bus Read Block	CR	
00110	0	0	101	Bus Read Block	CRI	
01000	0	0	001	Bus NC Read	NCR	

† x is a zero or one depending on what Mbus command is sent. Normally a CI command is sent. A CRI is only sent if the line was invalidated between the tag look up and when the command was sent on Mbus.

‡ A Flush command is sent to bcmd during a miss with owned sub-blocks and MXCC is bus owner. The Flush command is initiated from the Mbus side of the MXCC.

**B.III.5.4.7 Mbus Port Address Register**



Field	Description	Type
MDEV	MBus device number = 0x1	R
MREV	Device revision number = 0x0	R
MVEND	MBus vendor number = 0x4 (Texas Instruments)	R
MID<3:0>	MBus module ID (from Mbus connector)	R
rsvd	reserved, read as '0'	R



**B.III.5.4.8 BIST Register**

This register is used to trigger and monitor a Built In Self-Test (BIST) cycle. The BIST cycle is triggered by any processor write to this address; the IU will be suspended for approximately one second while BIST is run, and then can read the resulting signature in bits <30:0>. The signature value is TBD.

**B.III.6 MMU Details**

The MMU contains 64 fully-associative TLB entries with a limited-history LRU replacement algorithm. The page tables are cacheable in the E-\$ if MCR bit 16 is '1', but will never be cached in the D-\$.

The limited-history replacement algorithm first sweeps TLB entries sequentially until all TLB's have the valid bit set. From that point the next access to any TLB will set its 'used' bit; replacement cycles will take the first TLB that is not 'used'. At the time that all TLB entries have the 'used' bit set, all 'used' bits except the last one to be set are cleared, and all history is lost. The demap-all operation will clear all 'used' and 'valid' bits.

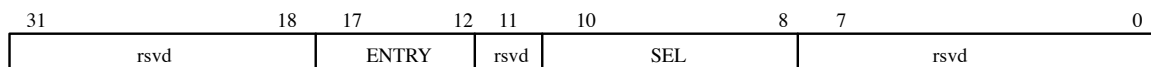
The root pointer and the level-2 PTP are also cached in the MMU. A context switch will invalidate the cached root pointer. Upon the first miss after a context switch, an extra level of table-walk is supported in order to fetch that context's root pointer. The level-2 PTP cache is invalidated upon writes to the context register or the context table pointer register, or upon table-walks that do not use that PTP (in which case a new level-2 PTP is obtained for the cache). This cached PTP is used only for table-walks, M-bit updates, and probe-entire operations. If level-2 is a PTE then it is not cached.

A table walk is *not* atomic on the MBus. Updates to the Modified and Referenced bits will guarantee correctness as follows: (1) if this is the first access to the page and it is not a store, the MMU will do an atomic swap operation to set the 'R' bit; if the swap return 'R' and 'M' both set (by another processor) then another swap is performed to set both 'R' and 'M'; (2) if this is a store access and the 'M' bit is not set in the PTE, a simple store is done with both 'R' and 'M' set.

Because hardware may be iteratively updating the PTE when software is attempting to write a new value or invalidate the entry, update algorithms must synchronize all processors and do a cooperative TLB flush in order to guarantee consistency.

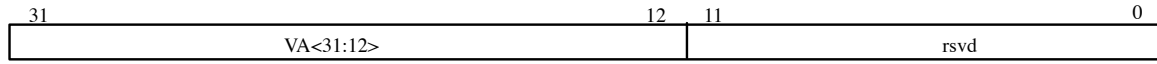
The Viking MMU implements the full set of PROBE types.

**B.III.6.1 TLB RAM Diagnostic Access Address Format**



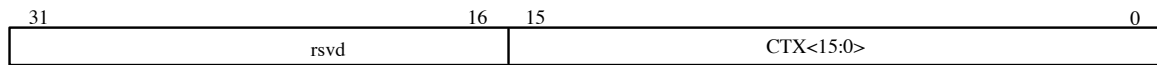
Field	Description
ENTRY SEL	Entry in Page Descriptor Cache (1 of 64) Selects portion of TLB to access. 0 = VA, 1 = Context, 2 = LOCK bit, 3 = PTE, 4 = Root Pointer (cached), 5 = level-2 PTP (cached), 6 = Level-2 Vaddr (cached)
rsvd	reserved: these bits are ignored

**B.III.6.1.2 VA Format (sel = 0 and sel = 6)**



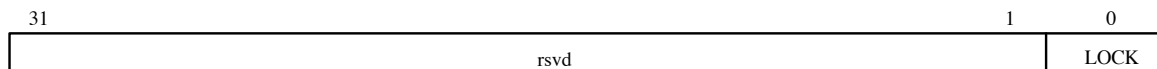
Field	Description	Type
VA<31:12>	Virtual address tag. Depending on PTE level, not all bits are significant.	RW
rsvd	reserved, read as '0'	R

**B.III.6.1.3 Context Format (sel = 1)**



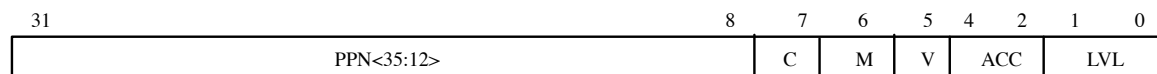
Field	Description	Type
CTX<15:0>	Context Tag.	RW
rsvd	reserved, read as '0'	R

**B.III.6.1.4 LOCK Format (sel = 2)**



Field	Description	Type
LOCK	If '1', the contents of this TLB entry will not be replaced	RW
rsvd	reserved, read as '0'	R

**B.III.6.1.5 PTE Format (sel = 3)**

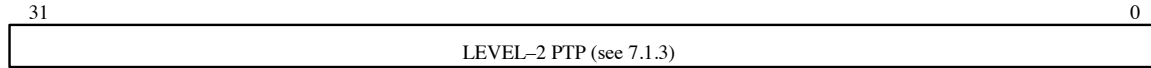


Field	Description	Type
PPN<35:12>	Physical Page Number	RW
C	Cacheable	RW
M	Modified	RW
V	Valid TLB entry	RW
ACC	Access Permission; same as SRMMU	RW
LVL	PTE level; 0 = root, 1 = region, 2 = segment, 3 = page.	RW

**B.III.6.1.6 Root Pointer Cache Format (sel = 4)**



**B.III.6.1.7 Level-2 PTP Cache Format (sel = 5)**



Field	Description	Type
<b>VFPA</b>	<b>Virtual Flush/Probe Address. Depending on type, not all bits may be significant.</b>	<b>RW</b>
<b>TYPE</b>	<b>Demapping type. 0 = page, 1 = segment, 2 = region, 3 = con-</b>	<b>RW</b>
<b>rsvd</b>	<b>text, 4 = entire MMU. 5-7 are reserved. reserved, writes are ignored.</b>	<b>R</b>
		<b>R</b>

**B.III.7 ASI's Implemented**

ASI	FUNCTION	SIZE
<b>0x2</b>	<b>MXCC registers and E-\$ access</b>	<b>Double</b>
<b>0x3</b>	<b>Ref MMU Flush/Probe</b>	<b>Word</b>
<b>0x4</b>	<b>Module Registers</b>	<b>Word</b>
<b>0x6</b>	<b>SRMMU Diagnostic I/D-TLB</b>	<b>Word</b>
<b>0x8</b>	<b>User Instruction</b>	<b>All</b>
<b>0x9</b>	<b>Supervisor Instruction</b>	<b>All</b>
<b>0xA</b>	<b>User Data</b>	<b>All</b>
<b>0xB</b>	<b>Supervisor Data</b>	<b>All</b>
<b>0xC</b>	<b>I-\$ Cache Tag</b>	<b>Double</b>
<b>0xD</b>	<b>I-\$ Cache Data</b>	<b>Double</b>
<b>0xE</b>	<b>D-\$ Cache Tag</b>	<b>Double</b>
<b>0xF</b>	<b>D-\$ Cache Data</b>	<b>Double</b>
<b>0x20-0x2F</b>	<b>SRMMU bypass, PA&lt;35:32&gt; = ASI&lt;3:0&gt;</b>	<b>All</b>
<b>0x30</b>	<b>Store Buffer Tags</b>	<b>Double</b>
<b>0x31</b>	<b>Store Buffer Data</b>	<b>Double</b>
<b>0x32</b>	<b>Store Buffer Control</b>	<b>Single</b>
<b>0x36</b>	<b>I-cache Flash Clear</b>	<b>Word</b>
<b>0x37</b>	<b>D-cache Flash Clear</b>	<b>Word</b>
<b>0x38</b>	<b>MMU Breakpoint Diagnostics (See Viking Specification for details)</b>	<b>Double</b>
<b>0x39</b>	<b>BIST Diagnostics (See Viking Specification for details)</b>	<b>Word</b>
<b>0x40-0x41</b>	<b>Emulation temps [1-2] (See Viking Specification for details)</b>	<b>Word</b>
<b>0x44</b>	<b>Emulation Data In1 (See Viking Specification for details)</b>	<b>Word</b>
<b>0x46-4C</b>	<b>Emulation Registers (See Viking Specification for details)</b>	<b>Word</b>

Alternate space accesses with unsupported ASI's will result in an error trap.

**B.III.8 Module Control Space Address Map**

PA<35:0>	Name
0xFFn000000 – 0xFF10FFFFFF	E-cache data
0xFFn800000 – 0xFF18FFFFFF	E-\$ tag
0xFFnC00000	Stream Data
0xFFnC00100	Stream Source
0xFFnC00200	Stream Destination
0xFFnC00300	Reference/Miss Count
0xFFnC00400 – 0xFFnC00700	Reserved ( <i>Not Used in MBus systems</i> )
0xFFnC00800	BIST
0xFFnC00900	Reserved
0xFFnC00A00	MXCC Control
0xFFnC00B00	MXCC Status
0xFFnC00C00	Module Reset
0xFFnC00D00	Reserved
0xFFnC00E00	Error Registers
0xFFnC00EFC – 0xFFnFFFFFFC	Reserved
0xFFnFFFFFFC	MBus Port Address Register

In this diagram 'n' represents the MID of the processor being accessed.

Control space accesses to the Stream registers do not interlock on pending operations as they do for processor-side accesses; the RDY bits must be polled. Also writes to the Stream Data Register can only happen when there is no processor-initiated stream operation in progress.

**B.III.9 IU PSR Number**

Bits <27:24> VER = 0x0

Bits <31:28> IMPL = 0x4 (Texas Instruments)

**B.III.10 Module-Specific Quirks**

This module violates the MBus specification for coherent snooping with MIH\* in cycle A+2; instead it invokes MBus Specification Appendix B.7, and provides MIH\* in cycle A+7.

Since this is a module with physically addressed caches there is no VA superset provided in the address phase of MBus transactions.

Note that locking all TLB entries can lead to a deadlock (infinite table walk) and so must be avoided.

**B.IV: Module Notes: Ross 604/64K**

*Based on Cypress Semiconductor 'SPARC RISC Users Guide' Second Edition, February 1990, with corrections based on conversations with Ross Semiconductor.*

- B.IV.1 Module Overview
- B.IV.2 Cache Details
- B.IV.3 Cache Flushing
- B.IV.4 Module Registers: differences from core (section 4)
- B.IV.5 Additional Registers Specific to this Module
- B.IV.6 MMU Details
- B.IV.7 ASI's Implemented
- B.IV.8 Module Control Space Address Map
- B.IV.9 IU PSR Number
- B.IV.10 Module-Specific Quirks
- B.IV.11 Module Write Buffers
- B.IV.12 Exiting Boot State

**B.IV.1 Module Overview**

The Ross 604 module consists of a chipset developed by Cypress Semiconductor/Ross Technologies. The processor set is a 1-scalar SPARC processor and cache system. The chipset includes the CY7C601 IU, a CY7C602 FPU, a CY7C604 CMU (Cache Controller and MMU) and two CY7C157 16KB x 16 pipelined cache SRAM. The entire chipset runs synchronous with the 40 MHz MBus clock. This is a level-1 module, meaning that it can only be used in a uniprocessor system. The MID for level-1 modules is hardwired to be 0xF, so it is not possible to use multiple level-1 modules in a system; this is due to hardware interlocks that use the MID, which is required to be unique for each master.

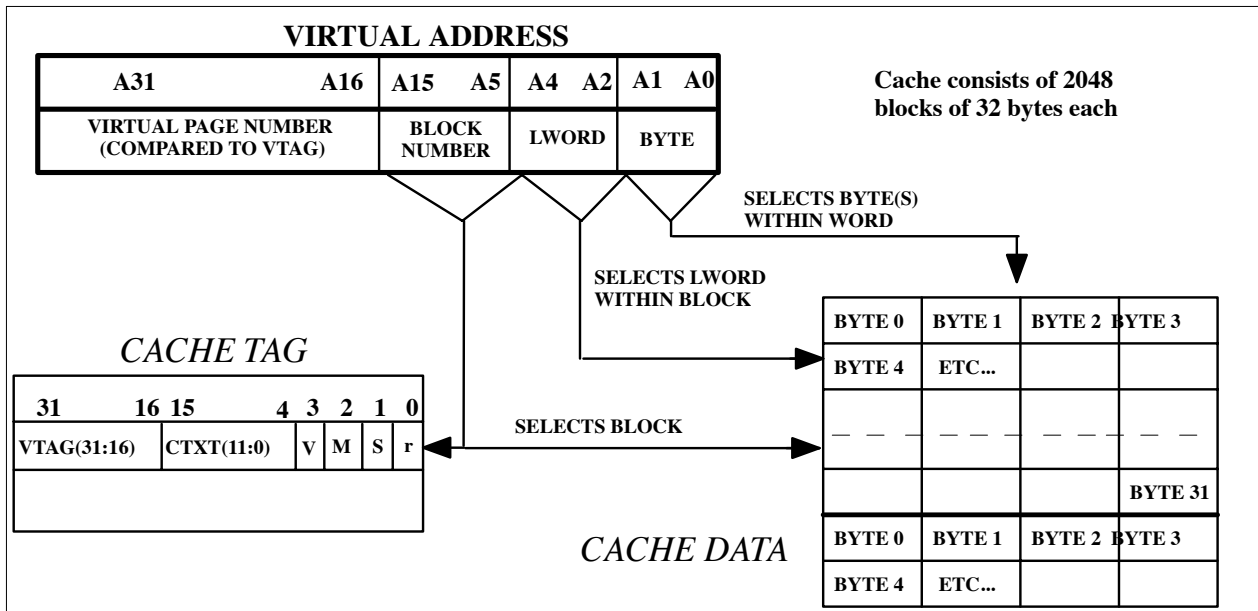
This logical module appears in one physical implementations, the Ross 6001 module (uniprocessor). This module is used only for system bringup purposes, and is not a product with Sun.

**B.IV.2 Cache Details**

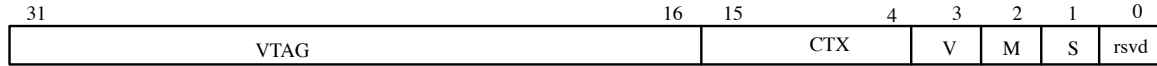
The Ross 604 system utilizes a 64 KB virtual-address direct-mapped cache, with one cache shared by both instruction access and data access. The cache does not participate in the MBus level-2 coherence protocols. The data cache can be programmed to be either a write-through cache with a no-write-allocate policy or a write-back cache with write-allocate. The latter is recommended for use in Sun-4M systems.

The cache consists of 2048 blocks of 32 bytes each. The tag for each block in the main directory contains the *virtual tag* VA<31:16>, the *context number*, a *valid* bit, a *modified* bit, and a *Supervisor* bit.

In write-back mode, aliases are treated by checking the physical address of the modified line being displaced against the physical address of the missed store access. If they match then the tag is updated and no memory activity is needed. Note that this check could require up to two table-walks to bring the physical addresses into the TLB's. Cache alias size is 64 KB.



**B.IV.2.1 Cache Virtual Tag Format**



Field	Description	Type
VTAG	Virtual Tag VA<31:18>	RW
CTX	Context Number CTX<11:0>	RW
V	Valid cache entry	RW
M	Modified cache line	RW
S	Supervisor mode	RW
rsvd	reserved: read as '0'	R

**B.IV.2.2 Addresses for Cache Diagnostic Access**

Cache Line	TAG (ASI = 0xE)	Data (ASI = 0xF)
0	0x00000000	0x00000000
1	0x00000020	0x00000020
2	0x00000040	0x00000040
3	0x00000060	0x00000060
...		
2047	0x0000FFE0	0x0000FFE0

**B.IV.3 Cache Flushing**

The cache in the Ross 604 module is flushed locally by the processor. Flushing involves use of the I/D-cache flush ASI's with *sta* accesses.

**B.IV.4 Module Registers: differences from core (section 4)**

**B.IV.4.1 Module Control Register: differences from core (section 4.1)**

- Bit <10>: CB, rw. Only copy-back mode is used in Sun-4M, so this bit should be set to '1'.
- Bit <19>: MV, rw. Multichip Valid; indicates that multiple 604's are tied together to provide a larger cache. In Sun-4M configurations this bit must be set to '0'.
- Bits <21:20>: MCM, rw. Multichip mask. *Not used in Sun-4M*
- Bits <23:22>: MCA, rw. Multichip address. *Not used in Sun-4M*
- Bits <27:24>: SRMMU VER = 0x1, ro; outdated prototypes have VER = 0x0
- Bits <31:28>: SRMMU IMPL = 0x1 (Cypress Semiconductor), ro.

**B.IV.4.2 Context Table Pointer Register: differences from core (section 4.2)**

The Context Table Pointer Register has bits <31:10> = CTP <35:14>, and bits <9:0> are reserved. This means that the context table must be 16KB-aligned in memory.

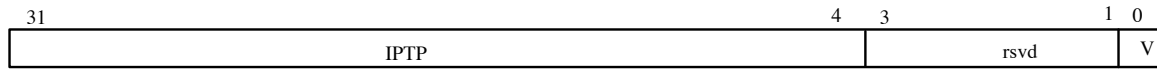
**B.IV.4.3 Context Register: differences from core (section 4.3)**

N = 11, thus 4096 contexts are supported.



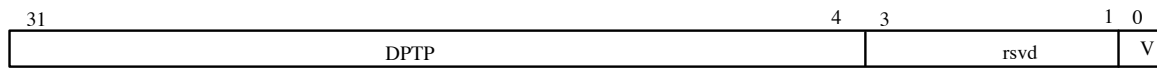


**B.IV.5.2 Instruction Access PTP Register**



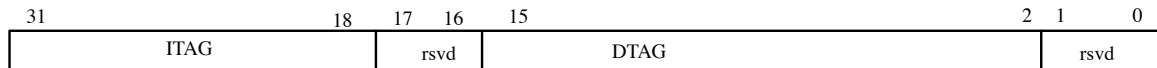
Field	Description	Type
<b>IPTP</b>	<b>Instnction Access PTP (page table pointer). Contains the level-2 PTP for IFETCH. It is part of the page table pointer cache. Valid. Cleared on power-on reset.</b>	<b>RW</b>
<b>V</b>	<b>reserved</b>	<b>RW</b>
<b>rsvd</b>		<b>R</b>

**B.IV.5.3 Data Access PTP Register**



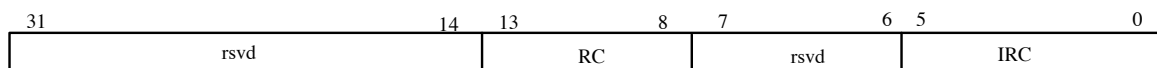
Field	Description	Type
<b>DPTP</b>	<b>Data Access PTP (page table pointer). Contains the level-2 PTP for Data access. It is part of the page table pointer cache. Valid. Cleared on power-on reset.</b>	<b>RW</b>
<b>V</b>	<b>reserved</b>	<b>RW</b>
<b>rsvd</b>		<b>R</b>

**B.IV.5.4 Index Tag Register**



Field	Description	Type
<b>ITAG</b>	<b>Tag for the IPTP register (level 1 and level 2)</b>	<b>RW</b>
<b>DTAG</b>	<b>Tag for the DPTP register (level 1 and level 2)</b>	<b>RW</b>
<b>rsvd</b>	<b>reserved</b>	<b>R</b>

**B.IV.5.5 TLB Replacement Control Register**



Field	Description	Type
<b>RC</b>	<b>Replacement Counter for TLB random replacement</b>	<b>RW</b>
<b>IRC</b>	<b>Initial Replacement Counter</b>	<b>RW</b>
<b>rsvd</b>	<b>reserved</b>	<b>R</b>

Both RC and IRC are reset to '0' upon power-on reset. In order to support TLB locking, the IRC can be set to a non-zero value. The IRC is used as an initialization value for RC; whenever RC reaches maximum count, it is pre-loaded with the value in IRC on the next increment. Locked TLB entries can be read/written through control space accesses (ASI = 0x6). When writing to the IRC field, write the same value to the RC field to ensure that the next-replacement pointer points to the unlocked area.

**B.IV.6 MMU Details**

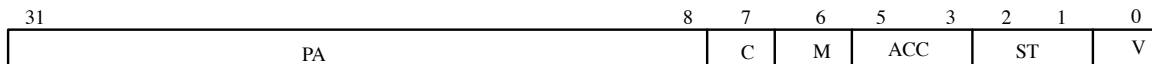
The MMU contains 64 TLB entries with a random replacement algorithm. The page tables are not cacheable and must be kept valid in main memory.

Upon the first miss after a context switch, an extra level of table-walk is supported in order to fetch that context's root pointer. A small cache keeps the most recently accessed PTP for the root level and one each for the level-2 PTP used for data and instruction access. This cache is visible as the RPR, IPTP, DPTP, and ITR registers. The IPTP and DPTP cache is flushed upon any TLB flush or upon any table walk for an instruction or data access, respectively. The entire PTP cache is flushed when the Context Register or the Context Table Pointer Register is written. A table walk is atomic, that is, MBSY\* is held asserted for the duration of a table-walk (up to 5 accesses to main memory in the case of a first-write to a page), and the LOCK bit in MBus address phase is asserted for table-walks.

The IPTP and DPTP are not updated during table walks caused by address alias detection or copy-back flushes.

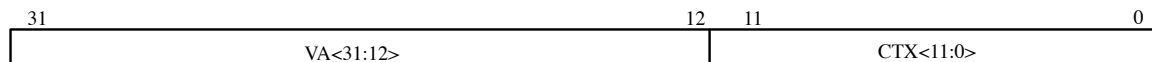
PROBE\_ENTIRE is the only probe type supported in this SRMMU implementation.

**B.IV.6.1 TLB RAM Diagnostic Access Format**



Field	Description	Type
V	Valid Entry	RW
ST	Short translation Bits. 0 = page level and 3 = root.	RW
ACC	Access Permission (per SRMMU)	RW
M	Modified; this page has been written to	RW
C	Cacheable page	RW
PA	Physical address <36:12>; some or all is valid depending on the ST field.	RW

**B.IV.6.1 TLB CAM Diagnostic Access Format**



Field	Description	Type
VA	Virtual Address tag for this entry	RW
CTX	Context tag for this entry	RW

**B.IV.7 ASI's Implemented**

Alternate space accesses with unsupported ASI's will be ignored (writes are ignored, reads provide garbage data).

This module has no hardware support for block copy or block fill.

Access with ASI = 0x1 is identical to access with ASI = 0x20 (Bypass, PA<35:32> = 0x0) with the exception that the MBL (local/boot mode) bit is asserted in the address phase. This bit is ignored in Sun-4M systems.

See the Ross 604 specification for details related to cache diagnostics, TLB diagnostics, TLB probes, cache flushing, and TLB flushing.

ASI	FUNCTION
0x1	Local Bus Mode (not used in Sun-4M)
0x3	Ref MMU Flush/Probe D-TLB
0x4	Module Registers
0x6	SRMMU Diagnostic I/D-TLB
0x8	User Instruction
0x9	Supervisor Instruction
0xA	User Data
0xB	Supervisor Data
0xE	D-\$ Cache Tag
0xF	D-\$ Cache Data
0x10	Flush I/D cache by page
0x11	Flush I/D cache by segment
0x12	Flush I/D cache by region
0x13	Flush I/D cache by context
0x14	Flush I/D cache by 'user'
0x20-0x2F	SRMMU bypass, PA<35:32> = ASI<3:0>

#### B.IV.8 Module Control Space Address Map

The CY7C604 has no MBus slave port.

#### B.IV.9 IU PSR Number

Bits <27:24> VER = 0x1 (Cypress Semiconductor)

Bits <31:28> IMPL = 0x1

#### B.IV.10 Module-Specific Quirks

The CY7C604 does not sense the MBus Module ID from the connector pins. Instead the MID is hardwired to 0xF; this is typical of level-1 modules.

The CY7C604 does not provide User/Supervisor information in the address phase of MBus transactions; for this reason any error status captured in system asynchronous error registers will always appear to be a supervisor-mode error, independent of the actual IU state.

The CY7C604 does *not* assert the module error signal upon watchdog reset since it is level-1.

The AFAR should be read only if AFV in the AFSR is asserted (see B.IV.4.5).

#### B.IV.11 Module Write Buffers

The CY7C604 contains 32-bytes of data write buffer. When the chip is programmed to be a copy-back cache, the write buffer is used to hold 1 cache write-back (32 bytes, one address) or 1-4 non-cacheable writes (up to 4 double-word stores and up to four addresses). The write buffer maintains strong order among writes issued, and reads or cacheable writes (coherent invalidates) will stall the IU if the write buffers are not idle.

A read of a non-cacheable entity (such as a hardware register, i.e. the M-to-S AFSR) will guarantee draining of such write buffers at context switch.

## **B.IV.12 Exiting Boot State**

### **B.IV.12.1 Exiting Boot State With MMU On**

The transition from boot mode to normal operation is a delicate time. In order to keep pipelines simple it is necessary to make the transition from boot-mode ifetch to instruction fetch translation (see 3.1.2) while executing from the same page; that is, the CY7C605 must be managed in a way that the same physical addresses are issued before, during, and after the change in the BM bit.

The physical address range for the EPROM is 0xFF000000 – 0xFF007FFFF. Address bits <23:19> are don't-care (they are not decoded by the EPROM interface) and so they can have any value. In boot-mode Ifetch pass-thru mode, the VA-to-PA translation is  $PA\langle 35:28 \rangle = 0xFF$ ,  $PA\langle 27:0 \rangle = VA\langle 27:0 \rangle$ . This means that  $VA\langle 31:28 \rangle$  are also don't-care when the module is in boot-mode. When boot mode is turned off, instruction fetches will be translated; the translation of this virtual address to the EPROM physical address must be established so that the fetches just before and just after the instruction that clears boot mode all fall on the same physical page. **Note that this means that  $PA\langle 27:24 \rangle = VA\langle 27:24 \rangle = 0x0$ ,  $PA\langle 18:0 \rangle = VA\langle 18:0 \rangle$ , and the rest of the address can be freely selected.**

### **B.IV.12.2 Exiting Boot State With MMU Off**

It is also legal to exit boot-mode with the MMU disabled; when boot-mode is turned off and the MMU is off, all accesses happen in MMU pass-thru mode (see 3.1.2) where  $PA\langle 35:32 \rangle = 0x0$ , and  $PA\langle 31:0 \rangle = VA\langle 31:0 \rangle$ . In order to guarantee correct operation, the page in which boot mode is turned off must be copied from the EPROM to main memory at an address where  $PA\langle 18:0 \rangle$  is identical to  $PA\langle 18:0 \rangle$  in the EPROM address; the transition from boot mode to pass-thru will also involve a transition from EPROM fetch to main memory fetch.

**B.V: Module Notes: Ross 605/128K**

*Based on Cypress Semiconductor 'SPARC RISC Users Guide' Second Edition, February 1990, with corrections based on conversations with Ross Semiconductor.*

- B.V.1      Module Overview
- B.V.2      Cache Details
- B.V.3      Cache Coherence
- B.V.4      Module Registers: differences from core (section 4)
- B.V.5      Additional Registers Specific to this Module
- B.V.6      MMU Details
- B.V.7      ASI's Implemented
- B.V.8      Module Control Space Address Map
- B.V.9      IU PSR Number
- B.V.10     Module-Specific Quirks
- B.V.11     Module Write Buffers
- B.V.12.    Exiting Boot State
- B.V.13     Programming Notes on Multichip Operation

**B.V.1 Module Overview**

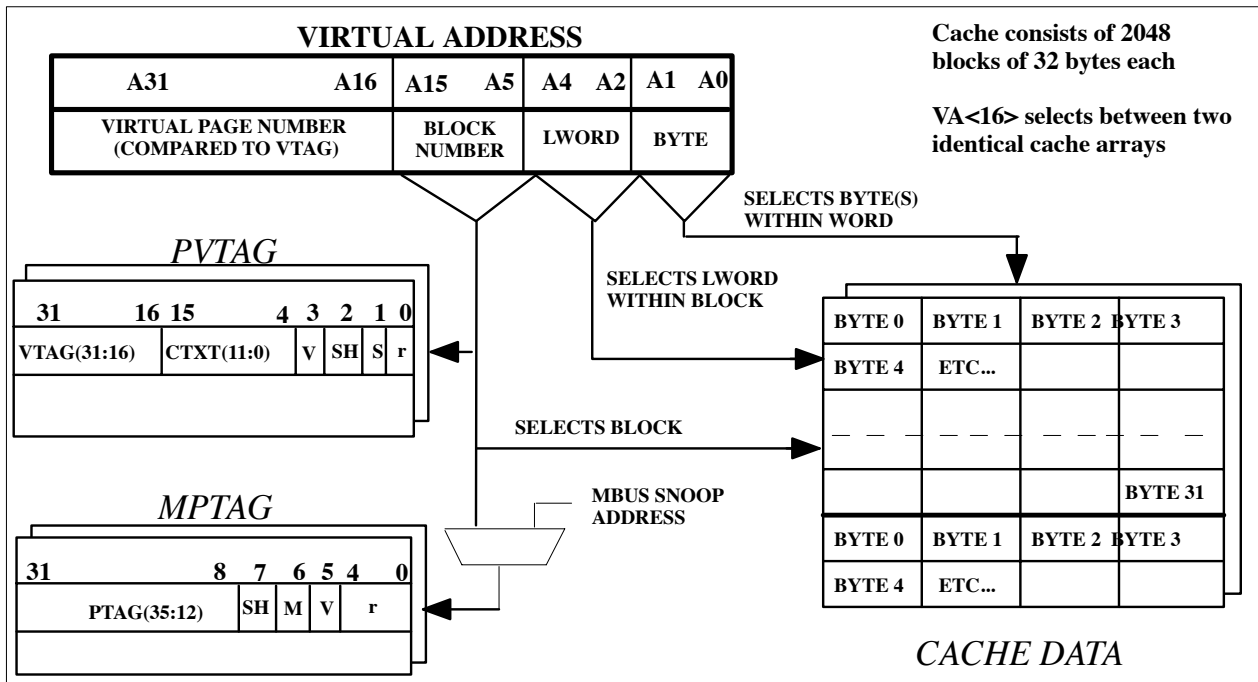
The Ross 605/128K module consists of a chipset developed by Cypress Semiconductor/Ross Technologies. The processor set is a 1-scalar SPARC processor and cache system. The chipset includes the CY7C601 IU, a CY7C602 FPU, two CY7C605 CMU (Cache Controller and MMU) and four CY7C157 16KB x 16 pipelined cache SRAM. The entire chipset runs synchronous with the 40 MHz MBus clock.

This module is very similar to the Ross605/64K module described in appendix B.II, except that a 128 KB cache is supported through the use of two cascaded CY7C605 chips. Each of these supports 64 KB of cache, differentiated by the value of VA<16>. This means that a single processor module has two (dependent) sets of Context and Context Table Pointer Registers, two independent Reference MMU's, two sets of Synchronous and Asynchronous Fault Status and Fault Address Registers, two Reset Registers, etc. This module does *not* conform to the generic core register addresses in section 4.

*This module is a hypothetical module; the functions described should work, but this module has never been built and tested at Sun. If built, a physical module would contain one uniprocessor logical module, since each processor requires two request/grant pairs in this design. There are no plans to use this module in a product.*

**B.V.2 Cache Details**

The Ross 605 system utilizes a 128 KB virtual-address direct-mapped cache, with one cache shared by both instruction access and data access. The cache participates in the MBus level-2 coherence protocols. Snooping is implemented via a virtually-indexed, physically tagged 'dual' directory. The data cache can be programmed to be either a write-through cache with a no-write-allocate policy or a write-back cache with write-allocate. The latter is used in Sun-4M systems.

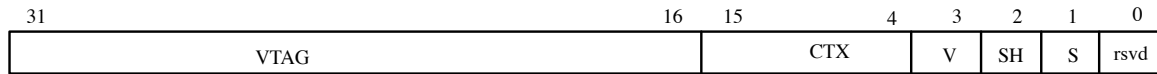


The cache consists of 4096 blocks of 32 bytes each. The tag for each block in the main directory contains the *virtual tag* VA<31:16>, the *context number*, a *valid* bit, a *shared* bit and a *Supervisor* bit. The 'dual' directory contains the physical tag for that block, along with a *valid*, *shared*, and *modified* bit.

In write-back mode, aliases are detected by checking the physical address of the miss with the 'dual' tag on the current occupant of the cache block. If an alias is detected then the miss can be serviced without accessing the system memory. **Cache alias size is 128 KB.**

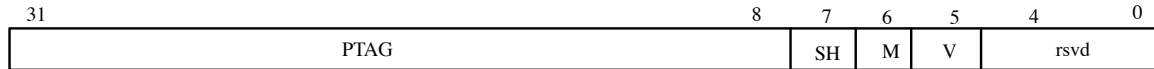
The 64 KB of cache supported by the first CY7C605 (referred to as CMU-0) are used for virtual addresses that have VA<16> = 0; the cache supported by the second CY7C605 (CMU-1) supports virtual addresses with VA<16> = 1. Normal cache operations (hits, misses, flushes) are serviced by the appropriate cache half, invisibly to software. Only control register operations need be aware of the dual-controller nature of this module.

**B.V.2.1 Cache Virtual Tag Format (PVTAG)**



Field	Description	Type
VTAG	Virtual Tag VA<31:18>	RW
CTX	Context Number CTX<11:0>	RW
V	Valid cache entry	RW
SH	Shared	RW
S	Supervisor mode	RW
rsvd	reserved: read as '0'	R

**B.V.2.2 Dual Directory Tag Format (MPTAG)**



Field	Description	Type
PT	Physical Tag PA<35:12>	RW
SH	Shared block	RW
M	Modified	RW
V	Valid cache entry	RW
rsvd	reserved: read as '0'	R

**B.V.2.3 Addresses for Diagnostic Access**

Cache Line	MPTAG (ASI = 0xE)		PVTAG (ASI = 0xE)		Data (ASI = 0xF)	
	605 #0	605 #1	605 #0	605 #1	605 #0	605 #1
0	0x00840000	0x01050000	0x00800000	0x01010000	0x00800000	0x01010000
1	0x00840020	0x01050020	0x00800020	0x01010020	0x00800020	0x01010020
2	0x00840040	0x01050040	0x00800040	0x01010040	0x00800040	0x01010040
3	0x00840060	0x01050060	0x00800060	0x01010060	0x00800060	0x01010060
...	...	...	...	...	...	...
2047	0x0084FFFE0	0x0105FFFE0	0x0080FFFE0	0x0101FFFE0	0x0080FFFE0	0x0101FFFE0

### B.V.3 Cache Coherence

#### B.V.3.1 Cache Flushing

The cache in the Ross 605 module is flushed locally by that module's processor. Flushing involves use of the cache flush ASI's with *sta* accesses. The dual-controller nature of the cache is not visible to software for flush operations.

#### B.V.3.2 Cache Snooping

The Ross 605 provides VA<19:16> and monitors VA<15:12> in the MBus VA Suerprset field; this is sufficient to snoop the 128 KB cache. **An important note is that there is no way to disable snooping, even when the cache is disabled; this means that all cache tags in all modules must be invalidated prior to enabling any of the caches in the system.**

#### B.V.4 Module Registers: differences from core (section 4)

Due to the fact that two CY7C605 parts are used, there are two complete sets of control registers. The address map for access to these registers differs from the generic core; in fact, **the generic core addresses cannot be used.** The 605/128K address map is as follows:

Address (ASI = 0x4)		Name
CMU-0	CMU-1	
0x00800000	0x01010000	Module Control Register (MCR)
0x00800100	0x01010100	Context Table Pointer Register (CTPR)
0x00800200	0x01010200	Context Register (CTX)
0x00800300	0x01010300	Synchronous Fault Status Register
0x00800400	0x01010400	Synchronous Fault Address Register
0x00800500	0x01010500	Asynchronous Fault Status Register
0x00800600	0x01010600	Asynchronous Fault Address Register
0x00800700	0x01010700	Reset Register
0x00801000	0x01011000	Root Pointer Register (RPR)
0x00801100	0x01011100	Instruction Access PTP (IPTP)
0x00801200	0x01011200	Data Access PTP (DPTP)
0x00801300	0x01011300	Index Tag Register (ITR)
0x00801400	0x01011400	TLB Replacement Control Register (TRCR)

#### B.V.4.1 Module Control Register: differences from core (section 4.1)

Bit <10>: CB, rw. Only copy-back mode is used in Sun-4M, so this bit should be set to '1'.

Bit <11>: MR, rw. Memory Reflection: should always be set to '0' in Sun-4M usage.

Bits <18:15>: MID<3:0>, rw. Ross 605 does not read the pins of the module connector to establish the MID, so the MID must be set by boot firmware. This must be done very early in the boot process (see B.V.10).

Bit <19>: MV, rw. Multichip valid (see B.V.11, 'Multichip Startup')

Bits <21:20>: MCM<1:0>, rw. Multichip mask (see B.V.11, 'Multichip Startup')

Bits <23:22>: MCA<1:0>, rw. Multichip address (see B.V.11, 'Multichip Startup')

Bits <27:24>: SRMMU VER = 0xF, (CY7C605), ro

Bits <31:28>: SRMMU IMPL = 0x1 (Cypress Semiconductor), ro.

Care must be taken with any operation that modifies the Module Control Registers. It is vital that any change in control information is made to both CY7C605's in a controlled way, such as enabling the cache, enabling the MMU, setting the 'C' bit or boot-mode bit, etc. The two module control registers will not be identically written; **once the multichip function is started, CMU-0 will have MCM = 0x2 and MCA = 0x2, and CMU-1 will have MCM = 0x2 and MCA = 0x3.** Both will have the MV bit set to '1' when multichip operations have been started. Any subsequent updates to the two module control registers must maintain the values of these fields.



**The MID value in the two CMU's will also differ.** The MID value needs to be determined prior to multichip operation (it should be done as soon as possible in the boot sequence). The number returned by the system MID Register should be an even number. CMU-0 gets this number, and CMU-1 gets that MID plus 1. For example, if the read of the MID Register returns '0x8', CMU-0 gets an MID of '0x8' and CMU-1 gets an MID of '0x9'. **This difference also must be maintained with each write to the module control registers.**

A power-on reset will turn off the MV bit in both MCT's, and will set the boot-mode bit in CMU-0; software reset through the Reset Register or Watchdog reset will set the boot-mode bit in both CMU's, and won't affect the current state of the MV bits. If multichip operation was started, then boot-mode accesses will be shared by both CMU's based on VA<16>.

#### **B.V.4.2 Context Table Pointer Register: differences from core (section 4.2)**

The Context Table Pointer Register has bits <31:10> = CTP <35:14>, and bits <9:0> are reserved. This means that the context table must be 16KB-aligned in memory. Both CTPR-0 and CTPR-1 must contain the same information; when one is updated, the other should be updated to the same value immediately.

#### **B.V.4.3 Context Register: differences from core (section 4.3)**

N = 11, thus 4096 contexts are supported. Both CTX-0 and CTX-1 must contain the same information; when one is updated, the other should be updated to the same value immediately.

#### **B.V.4.4 Synchronous Fault Status Register: differences from core (section 4.4)**

The SFSR is clear-on-read in this implementation. There is a set of synchronous fault status and address registers in each of the CY7C605's; when servicing a synchronous trap both SFAR-0, SFSR-0 and SFAR-1, SFSR-1 must be checked. Reading a particular SFSR clears it, and releases the corresponding SFAR.

#### **B.V.4.5 Asynchronous Fault Status Register: differences from core (section 4.5)**

Clearing of the AFSR is controlled by reads of the AFAR. There is a set of asynchronous fault status and address registers in each of the CY7C605's; when servicing an asynchronous fault (level-15 interrupt caused by a *module\_error*) both AFAR-0, AFSR-0 and AFAR-1, AFSR-1 must be checked. AFSR-0 and AFSR-1 should be read prior to reading AFAR-0 and AFAR-1, and the appropriate AFAR should be read only if AFV is asserted. This is done to prevent a race condition between asynchronous faults from the write buffers and accesses to the Asynchronous Fault registers.

#### **B.V.4.6 Reset Register: differences from core (section 4.6)**

This register is clear-on-read.

Bit<1> of CMU-1 resets only that MCT, and bit <1> of CMU-0 resets both CMU-0 and the IU/FPU. When issuing a software reset to this module, write first to Reset Register-1, then Reset Register-0. The first write will reset CMU-1, and the second write will reset CMU-0, the IU, and the FPU.

Snooping is maintained during either an SI or a WD reset.

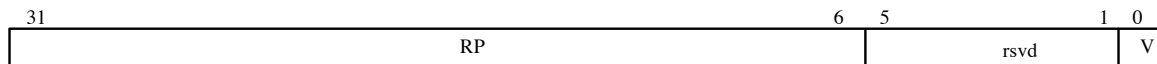
**B.V.4.7 MBus Port Address Register Register: differences from core (section 4.6)**

The CY7C605 does not support an MBus Port Address Register.

**B.V.5 Additional Registers Specific to this Module**

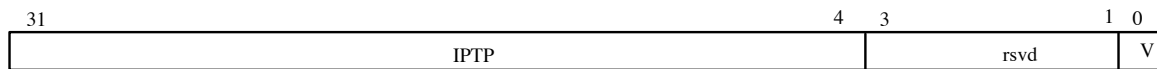
Address (ASI = 0x4)		Name
CMU-0	CMU-1	
0x00801000	0x01011000	<b>Root Pointer Register (RPR)</b> <b>Instruction Access PTP (IPTP)</b> <b>Data Access PTP (DPTP)</b> <b>Index Tag Register (ITR)</b> <b>TLB Replacement Control Register (TRCR)</b>
0x00801100	0x01011100	
0x00801200	0x01011200	
0x00801300	0x01011300	
0x00801400	0x01011400	

**B.V.5.1 Root Pointer Register**



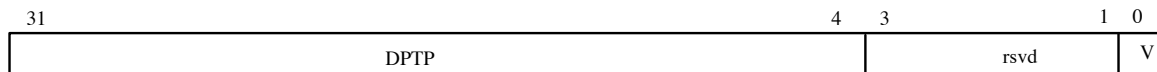
Field	Description	Type
RP	<b>Root Pointer:</b> This is the Context level table PTP (page table pointer). It is part of the page table pointer cache.	RW
V	<b>Valid.</b> Cleared on power-on reset, or on writes to the Context Register or to the Context Table Pointer Register.	RW
rsvd	<b>Reserved.</b>	R

**B.V.5.2 Instruction Access PTP Register**

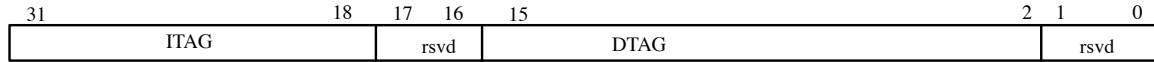


Field	Description	Type
IPTP	<b>Instruction Access PTP (page table pointer).</b> Contains the level-2 PTP for IFETCH. It is part of the page table pointer cache.	RW
V	<b>Valid.</b> Cleared on power-on reset.	RW
rsvd	<b>reserved</b>	R

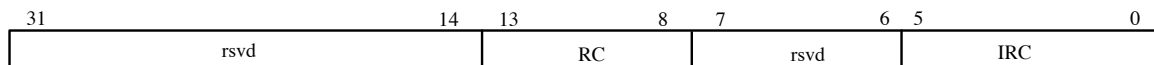
**B.V.5.3 Data Access PTP Register**



Field	Description	Type
DPTP	<b>Data Access PTP (page table pointer).</b> Contains the level-2 PTP for Data access. It is part of the page table pointer cache.	RW
V	<b>Valid.</b> Cleared on power-on reset.	RW
rsvd	<b>reserved</b>	R

**B.V.5.4 Index Tag Register**

Field	Description	Type
ITAG	<b>Tag for the IPTP register (level 1 and level 2)</b> <b>Tag for the DPTP register (level 1 and level 2)</b> <b>reserved</b>	<b>RW</b>
DTAG		<b>RW</b>
rsvd		<b>R</b>

**B.V.5.5 TLB Replacement Control Register**

Field	Description	Type
RC	<b>Replacement Counter for TLB random replacement</b> <b>Initial Replacement Counter</b> <b>reserved</b>	<b>RW</b>
IRC		<b>RW</b>
rsvd		<b>R</b>

Both RC and IRC are reset to '0' upon power-on reset. In order to support TLB locking, the IRC can be set to a non-zero value. The IRC is used as an initialization value for RC; whenever RC reaches maximum count, it is pre-loaded with the value in IRC on the next increment. Locked TLB entries can be read/written through control space accesses (ASI = 0x6). When writing to the IRC field, write the same value to the RC field to ensure that the next-replacement pointer points to the unlocked area.

**B.V.6 MMU Details**

The MMU contains 64 TLB entries with a random replacement algorithm. The page tables are not cacheable and must be kept valid in main memory.

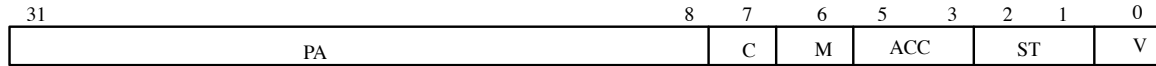
Upon the first miss after a context switch, an extra level of table-walk is supported in order to fetch that context's root pointer. A small cache keeps the most recently accessed PTP for the root level and one each for the level-2 PTP used for data and instruction access. This cache is visible as the RPR, IPTP, DPTP, and ITR registers. The IPTP and DPTP cache is flushed upon any TLB flush or upon any table walk for an instruction or data access, respectively. The entire PTP cache is flushed when the Context Register or the Context Table Pointer Register is written. A table walk is atomic, that is, MBSY\* is held asserted for the duration of a table-walk (up to 5 accesses to main memory in the case of a first-write to a page), and the LOCK bit in MBus address phase is asserted for table-walks.

The IPTP and DPTP are not updated during table walks caused by address alias detection or copy-back flushes.

This module supports two such MMU's; they are identical but independent. The MMU in CMU-0 responds to virtual addresses with VA<16> = 0, and the MMU in CMU-1 responds to virtual addresses with VA<16> = 1. When locking TLB entries this fact must be kept in mind.

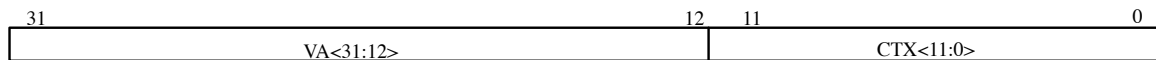
PROBE\_ENTIRE is the only probe type supported by this SRMMU implementation.

**B.V.6.1 TLB RAM Diagnostic Access Format**



Field	Description	Type
V	Valid Entry	RW
ST	Short translation Bits. 0 = page level and 3 = root.	RW
ACC	Access Permission (per SRMMU)	RW
M	Modified; this page has been written to	RW
C	Cacheable page	RW
PA	Physical address <35:12>; some or all is valid depending on the ST field.	RW

**B.V.6.2.TLB CAM Diagnostic Access Format**



Field	Description	Type
VA	Virtual Address tag for this entry	RW
CTX	Context tag for this entry	RW

**B.V.6.4 Addresses for TLB Diagnostic Access (ASI = 0x6)**

TLB Entry	TLB RAM		TLB CAM	
	CMU-0	CMU-1	CMU-0	CMU-1
0	0x00800000	0x01010000	0x00800004	0x01010004
1	0x00800008	0x01010008	0x0080000C	0x0101000C
2	0x00800010	0x01010010	0x00800014	0x01010014
3	0x00800018	0x01010018	0x0080001C	0x0101001C
...	...	...	...	...
63	0x008001F8	0x010101F8	0x008001FC	0x010101FC

**B.V.7 ASI's Implemented**

Alternate space accesses with unsupported ASI's will be ignored (writes are ignored, reads provide garbage data).

This module has no hardware support for block copy or block fill.

Access with ASI = 0x1 is identical to access with ASI = 0x20 (Bypass, PA<35:32> = 0x0) with the exception that the MBL (local/boot mode) bit is asserted in the address phase. This bit is ignored in Sun-4M systems.

See the Ross 605 specification for details related to TLB probes and TLB flushing.

ASI	FUNCTION
0x1	Local Bus Mode (not used in Sun-4M)
0x3	Ref MMU Flush/Probe I/D-TLB
0x4	Module Registers
0x6	SRMMU Diagnostic I/D-TLB ( <i>ref. B.V.6.4</i> )
0x8	User Instruction
0x9	Supervisor Instruction
0xA	User Data
0xB	Supervisor Data
0xE	I/D-\$ Cache Tag (A<18> = 0), MPTAG (A<18> = 1) ( <i>ref. B.V.2.3</i> )
0xF	I/D-\$ Cache Data ( <i>ref. B.V.2.3</i> )
0x10	Flush I/D cache by page
0x11	Flush I/D cache by segment
0x12	Flush I/D cache by region
0x13	Flush I/D cache by context
0x14	Flush I/D cache by 'user'
0x20-0x2F	SRMMU bypass, PA<35:32> = ASI<3:0>

### B.V.8 Module Control Space Address Map

The CY7C605 has no MBus slave port.

### B.V.9 IU PSR Number

Bits <27:24> VER = 0x1 (Cypress Semiconductor)

Bits <31:28> IMPL = 0x1

### B.V.10 Module-Specific Quirks

The CY7C605 does not sense the MBus Module ID from the connector pins. Instead the MID must be initialized in software. The MID can be read in the MID Register; this register is provided specifically for this module (see 5.4.3).

The CY7C605 does not provide User/Supervisor information in the address phase of MBus transactions; for this reason any error status captured in system asynchronous error registers will *always* appear to be a *supervisor-mode* error, independent of the actual IU state.

CY7C605 support for reflective memory and for second-level cacheing is not used in Sun-4M systems.

Due to the nature of module write buffers and the way that asynchronous errors are captured, the AFAR should not be read unless the corresponding AFV bit in the AFSR is asserted (see B.V.5.4)

### B.V.11 Module Write Buffers

The CY7C605's contains 32-bytes of data write buffer each. When the chip is programmed for multichip operation (as it is for this module) those write buffers are used only for copy-back operations; the buffering of non-cached stores is disabled to prevent store order problems between the two independent FIFO's. While the chip is still in single-chip mode the write buffer will hold either one 32-byte write-back or up to 4 non-cached stores. The software should ensure that the store buffers are drained prior to enabling multichip operation; this can be done by executing a non-cachable read.

## B.V.12. Exiting Boot State

### B.V.12.1 Exiting Boot State With MMU On

The transition from boot mode to normal operation is a delicate time. In order to keep pipelines simple it is necessary to make the transition from boot-mode ifetch to instruction fetch translation (see 3.1.2) while executing from the same page; that is, the CY7C605 must be managed in a way that the same physical addresses are issued before, during, and after the change in the BM bit.

The physical address range for the EPROM is 0xFF000000 – 0xFF007FFFF. Address bits <23:19> are don't-care (they are not decoded by the EPROM interface) and so they can have any value. In boot-mode ifetch pass-thru mode, the VA-to-PA translation is PA<35:28> = 0xFF, PA<27:0> = VA<27:0>. This means that VA<31:28> are also don't-care when the module is in boot-mode. When boot mode is turned off, instruction fetches will be translated; the translation of this virtual address to the EPROM physical address must be established so that several fetches just before and just after the instruction that clears boot mode all fall on the same physical page. **Note that this means that PA<27:24> = VA<27:24> = 0x0, PA<18:0> = VA<18:0>, and the rest of the address can be freely selected.**

In a multichip module the exit from boot state is done prior to enabling multichip operation, so the above activity involves a system where CMU-0 has the MMU enabled, CMU-1 is still MV = 0.

### B.V.12.2 Exiting Boot State With MMU Off

It is also legal to exit boot-mode with the MMU disabled; when boot-mode is turned off and the MMU is off, all accesses happen in MMU pass-thru mode (see 3.1.2) where PA<35:32> = 0x0, and PA<31:0> = VA<31:0>. In order to guarantee correct operation, the page in which boot mode is turned off must be copied from the EPROM to main memory at an address where PA<18:0> is identical to PA<18:0> in the EPROM address; the transition from boot mode to pass-thru will also involve a transition from EPROM fetch to main memory fetch.

## B.V.13 Programming Notes on Multichip Operation

### B.V.13.1 Notes on Multichip Startup

Prior to enabling multichip operation all instruction fetches are handled by CMU-0. When enabling multichip operation, it is very important the addresses used for those instructions live in an address space that is handled by CMU-0 (i.e. VA<16> = 0) in order to prevent conflict between the two CMU's during the period that one is multichip enabled and the other is not.

Whenever changing the state of the module control registers, the register belonging to CMU-1 should be written first, then the register belonging to CMU-0.

When enabling multichip operation, CMU-1 should have the module control register set with MCM = 0x2 and MCA = 0x3, and MV = 1. Next CMU-0 will have its module control register written with MCM = 0x2 and MCA = 0x2, and MV = 1. The same writes should initialize the MID fields as specified in B.V.10. The code should *not* immediately jump to addresses that have VA<16> = 1; to keep the pipelines uncomplicated and to prevent bus contention, the code should execute at least 5 instructions with VA<16> = 0 after multichip operation has been enabled.

For reference (in case the reader is referring to the CY7C605 data sheet) the 605's are designed in such way that they respond to different addresses depending upon whether or not the MV bit is set; the addresses chosen in this appendix are selected such that the 605's will respond correctly independent of the MV status.

### B.V.13.2 Handling Watchdog Resets and SE Resets with MV Valid

A watchdog reset will reset both CMU-0 and CMU-1, as well as the IU and FPU. The BM bit in both Module Control Registers will be set. The key difference between a WD reset and a POR reset is that the MV state of both CMU's is maintained; thus if multichip operation was enabled and a watchdog reset occurs, the bootmode instruction fetches will be handled by both CMU's as selected by VA<16>. **If the firmware chooses to disable multichip operation it is important to ensure that the VA being issued at that point in the code has VA<16> = 0 so that operation will continue when CMU-1 stops responding.** CMU-1 should have MV disabled first, then CMU-0 should have MV cleared so that CMU-0 will once again respond to all addresses.

SE resets (via the module Reset Registers) behave in a similar fashion.

### B.V.13.3 Probing 64K vs. 128K Modules

Once the boot code has determined from the IU PSR that this is a CY7C601 IU, it needs to determine if the module has a single CY7C604 (Appendix B.IV), a single CY7C605 (Appendix B.II) or a dual CY7C605 (this appendix). **This must be done prior to any other reads or writes to module control registers.**

The modules described in B.II and B.IV do not decode all address bits in ASI = 0x4 accesses, so the addresses used for CMU-0 and CMU-1 in this appendix will also access the registers in those modules. In order to determine which kind of module is in use, first read the Module Control Register using ASI = 0x4 and VA = 0x00800000. If the VER field = 0x0 this is a CY7C604-based module (appendix B.IV). If the VER field = 0xF then this is a CY7C605; in that case we must determine if this is a 64K or a 128K module. This is done by writing different data to the CTX registers CTX-0 (at VA = 0x00800200) and CTX-1 (at VA = 0x01010200) then reading them back. If the last data written appears at both addresses then this is a 64K module (appendix B.II), and if the two patterns persist this is a 128K module (this appendix).

In order to simplify the firmware, it is safe to use CMU-0 addresses in place of the Section 4 generic module addresses for ASI = 0x4 accesses on all CY7C601-based modules; all three module types listed above will respond correctly.