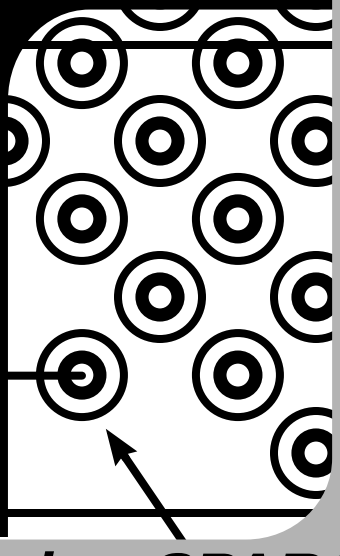
SUN MICROELECTRONICS



microSPARC™-IIep

User's Manual

June 1999

Sun microsystems

microSPARC™-IIep User's Manual

THE NETWORK IS THE COMPUTER™

Sun Microsystems, Inc.
Microelectronics
901 San Antonio Road
Palo Alto, CA 94303-4900 USA
800 / 681-8845
http: / / www.sun.com / microelectronics

Part No.: 802-7100-02
June 1999

Contents

Figures

Tables

Document Revision History

Date	Document Revision	Description of Change
June 1999	-02	revised to describe CPU part number 802-7327-03
April 1997	-01	initial draft; described CPU part number 802-7327-01

Preface

The microSPARC™-IIep is an extension of the SPARC™ processor family targeted for low-cost applications. The microSPARC-IIep RISC processor allows systems designers to take advantage of a highly integrated SPARC “system-on-a-chip” and achieve industry-leading performance.

The microSPARC-IIep integrates a 32-bit SPARC processor with floating-point unit, memory management unit, separate instruction and data caches, PCI bus controller, DRAM and flash memory controller, and clock generator using phase-locked loop on to a single device. Implemented with state-of-the-art CMOS technology, the microSPARC-IIep provides an ideal low-cost, high-performance, and low-power-consumption solution.

Like all SPARC processors, microSPARC-IIep processors are supported by the industry’s largest installed base of native RISC development environments, applications, and support tools. SPARC is the leading microprocessor technology supporting the information superhighway infrastructure in terms of hardware and software. These tools and technology make SPARC ideal for your embedded and networked computing applications.

Refer to the contents of the device ID register (see *Figure 11-4* on page 229) for the version of the microSPARC-IIep covered by this manual.

CHAPTER 1

microSPARC-IIep Overview

1.1 Introduction

The microSPARC-IIep CPU is a highly integrated, low-cost implementation of the SPARC version 8 RISC architecture with a PCI interface. Its implementation evolved from Sun's microSPARC architecture.

- High performance is achieved by the high level of integration, including on-chip instruction and data caches, built-in DRAM controller, and PCI local bus controller.
- A full-custom implementation allows for a target frequency of 100-133MHz providing sustained performance.
- The design is highly testable with support of full JTAG scan.
- The microSPARC-IIep chip supports up to 256 megabytes of DRAM and 4 external PCI slots.
- The ability to operate as either a PCI host master or intelligent PCI slave interface. Intelligent PCI slave mode implies that an external arbiter is used and that the microSPARC-IIep CPU is not the source of the PCI clocks or the PCI reset. Intelligent PCI slave mode is entered whenever the pll_byp_l pin is high and the ext_clk2 pin is high. If either of these pins are low, then the microSPARC-IIep CPU is in PCI host master mode. PCI configuration read/write commands are supported only in intelligent PCI slave mode.

Table 1-1 summarizes the key differences between the microSPARC-IIep CPU and the microSPARC-II CPU.

Table 1-1 Feature Comparison of the microSPARC-II CPU and the microSPARC-IIep CPU

Feature	microSPARC-II CPU	microSPARC-IIep CPU
Overall	• 32-bit SPARC Architecture version 8	
	• Supports big-endian byte ordering	• Supports little- and big-endian byte ordering
Frequency	• 110 MHz	• 100 MHz - 133 MHz
Integer Unit	• 136-word register file with 8 windows and 8 global registers • 5-stage pipeline • Supports branch folding • 4-deep instruction queue supporting instruction prefetching • Support instruction and data cache streaming	
	• Supports big-endian byte ordering	• Supports little- and big-endian byte ordering
Floating-Point Unit	• Supports all single- and double-precision floating-point SPARC version 8 instructions • Traps all quad-precision floating-point instructions • Datapath contains Meiko floating-point engine, fast multiply unit • Supports simultaneous execution of fast multiplications and other floating-point operations such as floating-point add • 3-entry floating-point deferred trap queue • 32 floating-point registers of 32 bits wide	
Memory Management Unit	• SPARC version 8 Reference MMU • Translates 32-bit virtual address to 31-bit physical address • Supports 8 different 256 MB address spaces • Supports 256 contexts	
	• 64-entry fully-associative TLB with pseudo random replacement algorithm	• 32-entry fully-associative TLB with pseudo random replacement algorithm
	• Unified memory TLB and IOTLB	• Separate memory TLB and IOTLB
	• Supports hardware table-walks	
Data Cache	• 8K-Byte, direct-mapped, virtually-indexed, virtually-tagged, write-through with write-allocate • 512 lines of 16 bytes • 4-deep write buffer of 64 bits wide	
Instruction Cache	• 16K-Byte, direct-mapped, virtually-indexed, virtually-tagged • 512 lines of 32 bytes	
Graphics Bus Interface	• High-speed local bus	• Not supported
Memory Interface	• Programmable DRAM controller • Supports up to 25 MB of system memory • 64-bit data and 2-bit parity • 8 RAS lines • 4 CAS lines • Supports 2 pages at a time • Supports 5V/3V standard/slow refresh, self-refresh	
	• Supports fast-page mode DRAM only	• Supports FPM or EDO DRAM that meets fast-page mode timing

Table 1-1 Feature Comparison of the microSPARC-II CPU and the microSPARC-IIep CPU *(Continued)*

Feature	microSPARC-II CPU	microSPARC-IIep CPU
Local Bus Controller	SBus	• PCI revision 2.1 • 32-bit, 33 MHz • Supports (in host mode) up to 4 external bus masters or slaves • Supports (in host mode) bus arbitration between host and 4 external masters • Supports host and satellite modes • Address translation from 32-bit local bus address to main memory space assisted by dedicated 16-entry IO TLB • Supports little- and big-endian byte ordering with automatic endian conversion • Supports direct transactions between external master and slave devices. DMA sustained write bandwidth is 70 Mb/s and sustained read bandwidth is 45 Mb/s • Supports PIO between the microSPARC-IIep CPU and PCI devices. Peak PIO write bandwidth is 70 Mb/s PIO read bandwidth is 25 Mb/s • Interrupt controller with programmable priority assignments and programmable output pins • PCI clock input generates the processor clock (x3 or x4) • Two 32-bit or one 32-bit and one 64-bit timers
Flash Memory Interface	• Not supported	• Supports 8-bit or 32-bit interface • Pin-selectable boot choice from 8 or 32-bit flash memory or from one of two PCI memory spaces • Industry-standard device interface (28FxxxXX) and support for industry standard programming algorithm
Boundary Scan JTAG		• Implements CLK_RST instruction to reset the clk_cntl block to a known state
TAP Controller		
Packaging	• 321-pin, pin grid array	• 272-pin, plastic ball grid array
Performance		• 72 SPECint92 • 59 SPECfp92 • 208K Dhrystone
Voltage		• Core operating voltage of 3.3V with preserved 5 V I/O compatibility

1.2 microSPARC-IIep Memory Map

The microSPARC-IIep physical memory address mapping is shown in Appendix B, *Physical Memory Address Map*.

1.3 microSPARC-IIep Endian Support

The microSPARC-II CPU works only with big-endian data but it includes endian conversion logic that allows it to handle either big or little-endian data. The endian conversion logic byte-swaps external little-endian data to convert it to internal big-endian data and correspondingly converts internal big-endian data to external little-endian data when appropriate. The endian conversion logic is enabled by bits 15 and 16 of the processor state register (PSR) and bit 2 of the PCI controller PIO control register.

1.3.1 Processor-internal Endian Support

The microSPARC-IIep CPU supports little-endian system memory data for both supervisory and user modes. PSR bits 16 and 15 enable little-endian conversion during supervisor and user modes, respectively:

- PSR [16]: When set, the default byte ordering for supervisor data references is little endian. When clear, the default byte ordering for supervisor data references is big endian.
- PSR[15]: When set, the default byte ordering for user data references is little endian. When clear, the default byte ordering for user data references is big endian.

For the following examples see *Figure 1-1*.

- Processor operating in little-endian mode: If the contents of a double word register (r2,r3) = 0001.0203.0405.0607 and a double word store to memory location 0 is issued, the double word at memory location 0 contains 0706.0504.0302.0100 after the transfer.
- Processor operating in big-endian mode: If the same double word register is transferred to memory location 0 while operating in big-endian mode, the double word at memory location 0 contains 0001.0203.0405.0607 after the transfer.

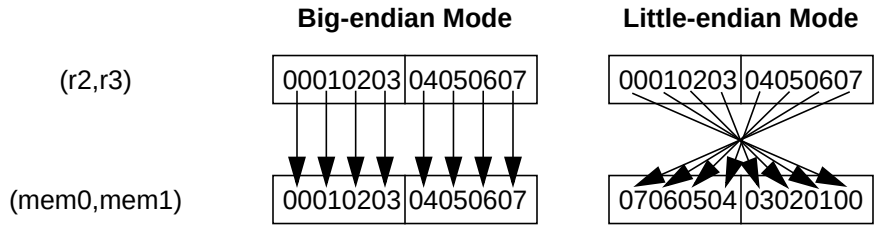


Figure 1-1 Big-endian vs. Little-endian Example (Processor Double Word Store)

All loads and stores, including ones using address space identifiers (ASI), are handled in the mode designated by the PSR endian control bit.

Note – Be sure to check the PSR endian control bits while performing maintenance operations in little-endian mode. Failure to do so may result in erroneous failure indications because the data may appear to be scrambled.

Switching of endian modes does not take effect until after completion of the instruction immediately following the PSR endian control bits update. When switching between endian modes, the instruction following the PSR modification will operate in the previous endian mode.

Note – When switching modes, the software must include a NOP, non-memory, or ASI shadow instruction following an update to the PSR (see *Figure 1-2*).

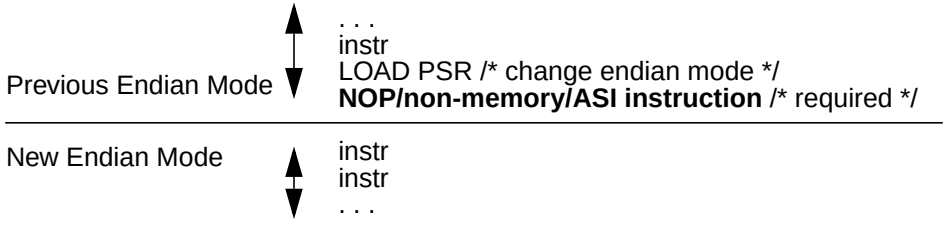


Figure 1-2 Required Shadow Instruction at Processor Endian Mode Switch

Caching of data is allowed while operating in little-endian mode, but there is no hardware mechanism in the data cache to determine if a particular datum is stored in big or little-endian format. The endian mode of the cached data is determined by the context identity value of the process. By tracking the context identity, the user can determine the endian mode of the cached data.

Note – Certain hardware operations of the microSPARC-IIep processor assume the byte ordering of the data references to be big endian only. For example, independent of the PSR settings, the data references for table walks are treated as big-endian data.

There is no performance penalty while operating in little-endian mode.

1.3.2 Processor External PIO Endian Support

The endian conversion logic across the processor-to-PCI interface is controlled by bit 2 of the PCI controller PIO control register (PA=0x300C.0060). On reset, the endian conversion logic is enabled. Therefore, data on the PCI bus is little-endian.

For the following example see *Figure 1-3*.

- If the processor is operating in big-endian mode, has contents 0001.0203.0405.0607 in the double word register (r2,r3), and bit 2 is set to 0, then a PIO initiated PCI memory write places the data 0302.0100 and 0706.0504 on the PCI bus in consecutive transactions.
- With bit 2 set to 1, no byte twisting is done. If the processor is operating in big-endian mode with bit 2 set and has contents 0001.0203.0405.0607 in the double word register (R2,R3), then a PIO initiated PCI memory write places the data 0001.0203 and 0405.0607 on the PCI bus in consecutive transactions.

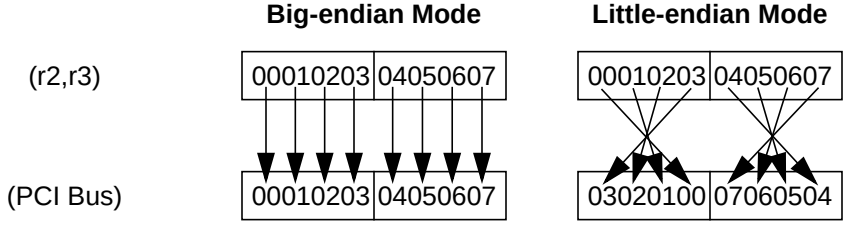


Figure 1-3 Big Endian vs. Little-endian Example (PCI Master Double Word Transfer)

This bit can be changed on the fly. The preferred method is to set the bit to the desired value, then read it back. This guarantees that other PIO transactions are locked out while the PIO endian control is in transition (see *Figure 1-4*).

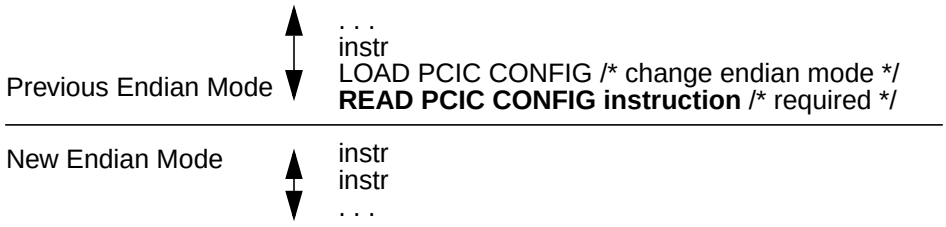


Figure 1-4 Required Readback Instruction at PCI Master Endian Mode Switch

1.3.3 DMA

The iafx_master / PCI slave (DMA PATH) contains no control bit. Data is always transferred between PCI memory and system memory in sequential byte order (PCI memory byte 0 is mapped to system memory byte 0, byte 1 is mapped to byte 1 and so on).

1.3.4 Settings for Endian Conversion

The following two sections describe the recommended register settings for the microSPARC-IIep CPU to operate in big and little-endian environments.

1.3.4.1 Big-endian Environment

PSR[16] and PSR[15] are both cleared to 0. PCI controller PIO control register [2] is set to 1. See *Table 1-3* for an example.

Table 1-2 Big-endian Example

Location	Data
microSPARC-IIep register	r2 r3 00010203 04050607
System memory	addr 0 7 data 00010203 04050607
PCI local bus	AD 31 0 CBE 3 0 data 00010203 AD 31 0 CBE 3 0 data 04050607

1.3.4.2 Little-endian Environment

PSR[16] and PSR[15] are set to 1 depending on whether data access is in supervisor or user mode. PCI controller PIO control register [2] is set to 0. See *Table 1-3* for an example.

Table 1-3 Little-endian Example

Location	Data
microSPARC-IIep register	r2 r3 00010203 04050607
System memory	addr 7 0 data 00010203 04050607
PCI local bus	AD 31 0 CBE 3 0 data 03020100 AD 31 0 CBE 3 0 data 07060504

1.4 Block Diagram

Figure 1-5 shows a typical microSPARC-IIep system block diagram.

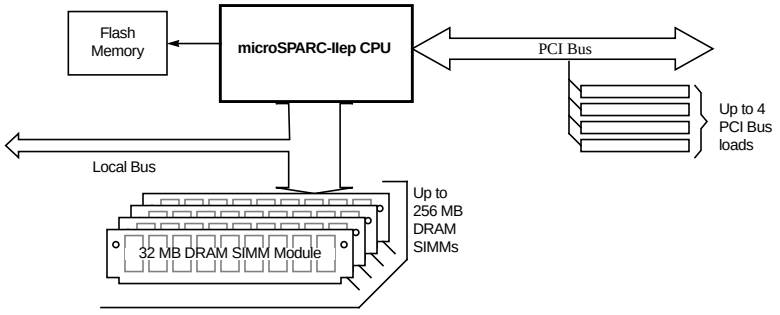


Figure 1-5 Typical microSPARC-IIep System Block Diagram

Figure 1-6 shows the microSPARC-IIep:

- Integer unit (IU)
- Floating-point unit (FPU)
- Instruction and data caches
- Memory management unit (MMU) with 32-entry translation lookaside buffer (TLB)
- DRAM controller
- PCI controller
- PCI bus interface
- IOMMU with 16-entry IOTLB
- Flash memory interface
- Interrupt controller
- Two timers
- Internal and boundary scan JTAG interface
- Power management
- Clock generation—using incoming PCI clock

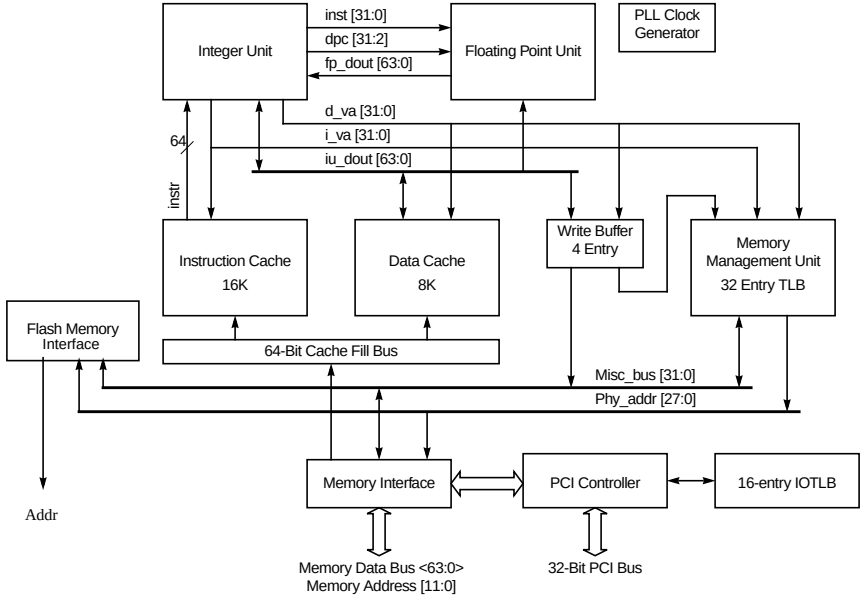


Figure 1-6 microSPARC-IIep Block Diagram

Operation requires an on-chip clock generator with a phase locked loop. The microSPARC-IIep processor operates at a multiple of the 33 MHz PCI bus input frequency, The clock generator multiplies the PCI input clock by six (200 MHz) or by eight (266 MHz) before dividing by two to generate the processor reference clock (100 MHz or 133 MHz).

Memory interfacing is done using four major buses:

The MEMIF block interfaces with the DRAM via a “b_memdata<63:0>” bus. The bus is divided into b_memdata_in<63:0> and b_memdata_out<63:0> inside the chip for input and output, respectively.

Cache_fill bus: a 64-bit unidirectional bus from b_memdata_in bus to the I-cache and D-cache. This bus is used for cache fill writes, bypass streaming and noncached load/fetches from DRAM.

Misc_bus (Miscellaneous bus): a 32-bit bidirectional bus, used for ASI load and store transfers; and write buffer for MEMIF transfers.

IAFX bus: an internal version of the AFX bus, enhanced to allow an external master to request the use of the bus and initiate memory transfers.

Figure 1-7 shows the microSPARC-IIep pipeline.

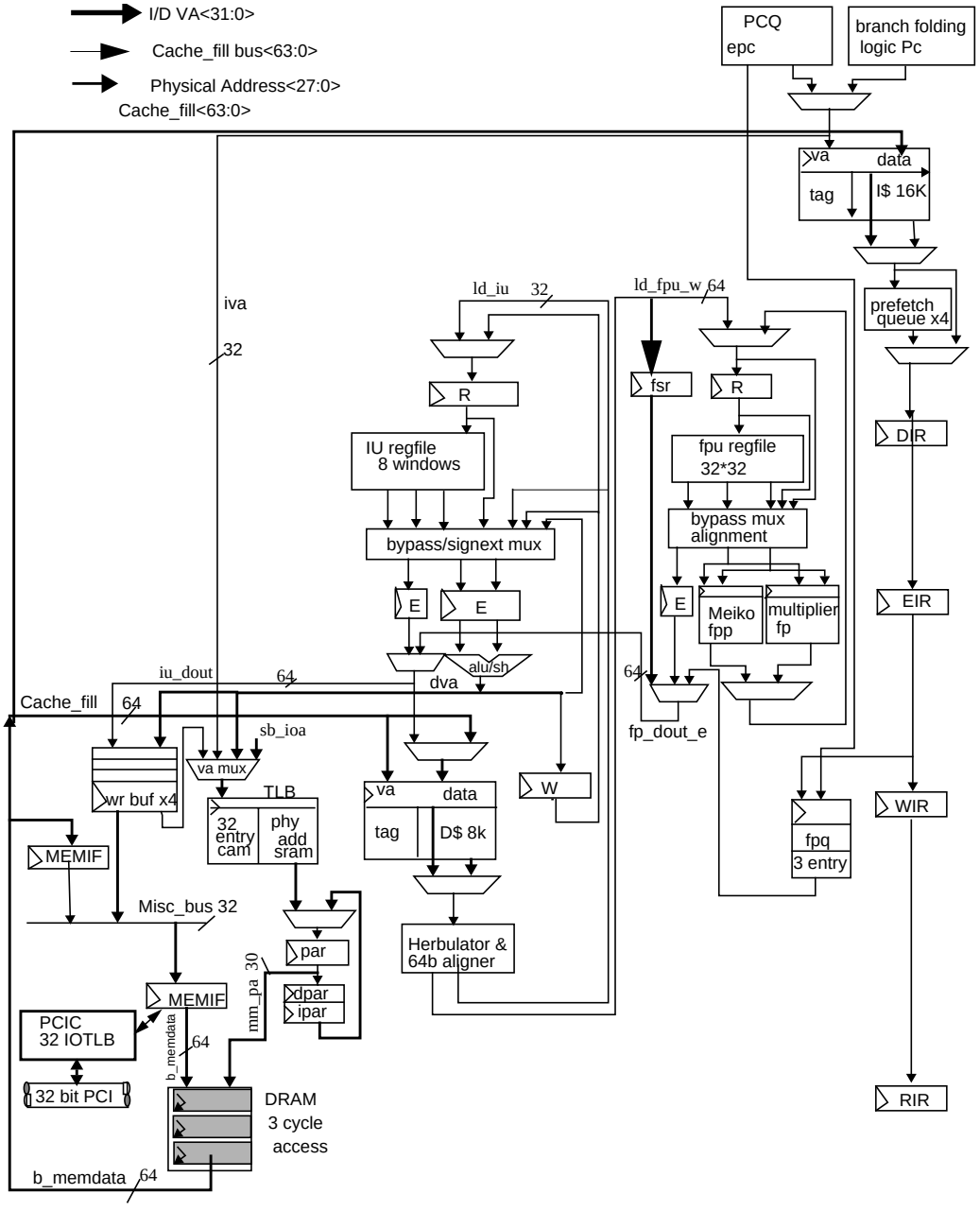


Figure 1-7 microSPARC-IIep Pipeline Diagram

CHAPTER 2

CPU Performance

2.1 Benchmark Configurations and Results

Table 2-1 through *Table 2-6* give results for the standard microSPARC II CPU. Differences that apply to the microSPARC-IIep CPU are explained following these tables.

The results of these benchmark tests at 100 MHz on microSPARC-II machines are presented in *Table 2-1*.

With a 32-entry TLB—compared with a 64-entry TLB for the microSPARC-II— the microSPARC-IIep CPU's performance is typically lower by 4.5% for SPECint92 and by 0.9% for SPECfp92 compared with these microSPARC-II results.

Note – Performance of programs that overflow the available TLB entries is lower than listed.

Table 2-1 microSPARC-II CPU Performance Summary

Benchmark	100 MHz
SPECint92	72.29
SPECfp92	59.28
Dhrystone	208.33K
MIPS	118.57
MFLOPS	8.89

2.1.1 Benchmark Test Configuration

The benchmark test setup is listed in *Table 2-2*.

Table 2-2 Benchmark Test Configuration

	Item	Configuration
Hardware	Model Number	SPARCstation 5-100
	CPU	100 MHz microSPARC II
	FPU	Integrated
	Number of CPUs	1
	Primary Cache	16 KB instruction + 8 KB data on chip
	Other Cache	None
	Memory	64 MB
	Disk Subsystem	1 GB single-ended SCSI
Software	Compilers	Apogee 3.051
	Other Software	Kuck & Associates KAP
	File System	UFS
	System State	Single User

2.1.2 SPECint92 Test Results

The SPECint92 test results are presented in *Table 2-3*. SPECint92 computed by best runs is 72.29 and SPECrate_int92 is 1864.

Table 2-3 Test Results for SPECint92

Benchmark	Copies	Elapsed Time	Best Runs
008.espresso	1	35.70	70.28
022.li	1	87.00	75.82
023.eqntott	1	7.80	154.93
026.compress	1	70.80	41.22
072.sc	1	36.80	135.22
085.gcc	1	117.90	51.12

2.1.3 SPECfp92 Test Results

The SPECfp92 test results are presented in *Table 2-4*. SPECfp92 computed by best runs is 59.28.

Table 2-4 Test Results for SPECfp92

Benchmark	Copies	Elapsed Time	Best Runs
013.spice2g6	1	542.90	44.21
015.doduc	1	37.10	50.13
034.mdljdp2	1	97.70	72.57
039.wave5	1	99.50	37.19
047.tomcatv	1	44.80	59.15
048.ora	1	79.30	93.57
052.alvinn	1	68.20	112.76
056.ear	1	285.00	89.47
077.mdljsp2	1	81.40	41.15
078.swm256	1	311.90	40.72
089.su2cor	1	207.40	62.20
090.hydro2d	1	298.00	45.97
093.nasa7	1	245.80	68.35
094.fpppp	1	152.20	60.45

2.1.4 Dhrystone Test Results

This machine benchmarks at 208,333 Dhrystone/second.

The performance projections for the microSPARC-IIep CPU are extrapolated from the actual performance figures for the microSPARC-II. This extrapolation is possible because the microSPARC-IIep CPU is based on the design of the microSPARC-II core. There are, however, minor differences in the I/O subsystems and the translation lookaside buffers (TLB) of the microSPARC-II CPU and the microSPARC-IIep CPU (that is, the microSPARC-II CPU has a 64-entry TLB with 16 entries dedicated for IOTLB use, while the microSPARC-IIep CPU has a 32-entry TLB). As a result, adjustments have to be made to the microSPARC-II data to account for these differences.

2.2 Compiler Optimization Guidelines

This section explains some of the code scheduling issues that affect the performance of the microSPARC-IIep processor.

2.2.1 Branches

Integer branches are either folded with their delay slot instructions or allowed to enter the integer pipeline.

Branch folding is supported by a four-deep instruction queue. The queue is filled each cycle by a double word fetch. For a branch to be folded, the branch, delay slot, and delay slot+1 instructions must be in the queue or be streaming to the integer unit (IU) from the instruction cache. In addition, the instruction preceding the branch cannot be a multi-cycle instruction or a control transfer instruction (CTI), and there cannot be a WRspec (write to a special register) in the pipe.

All branches are predicted taken. The target instruction is fetched in the D-stage of the delay slot instruction (or branch-delay slot pair).

```
   bicc 1f
   delay
   delay+1
   ...
1: target
   ...
```

Table 2-5 summarizes the cycles taken for a branch.

Table 2-5 Cycles for a Branch

Branch	Taken	Not Taken
Folded	0	1
Not Folded	1	1 or 2

If the branch can be folded, the branch and delay slot are executed at cycle x. If the branch is taken, the target executes at cycle x+1. If the branch is not taken, the target must be killed and delay+1 is executed at cycle x+2. Thus, folded taken branches take zero cycles, while folded untaken branches take one cycle.

If the branch cannot be folded, it enters into the pipeline at cycle x, and the delay slot instruction enters at cycle x+1. If the branch were taken, the target executes at cycle x+2. If the branch were not taken, but the delay instruction+1 is in the instruction queue, it executes at cycle x+3; otherwise it must be fetched and executes at cycle x+4.

2.2.2 Guidelines for Branch Folding

- Try to make as many BICC instructions taken as possible since the microSPARC-IIep CPU always predicts taken and fetches the target. If the branch is untaken, it costs a cycle if it were folded, and may cost an additional cycle if it were not folded.
- Avoid BICC to BICC control transfers. The target BICC cannot be folded since delay+1 will not be in the instruction queue.

```
        bicc    1f
        delay
        ...
1:      bicc2f
        ...
```

- Try to have CTI target instructions be double-word aligned (for example, label 1 is a double-word address). This allows the odd word to enter the queue immediately. If the odd word happens to be a BICC, it can be folded. If the target is an odd word, the following BICC does not enter the queue and is not be folded.

```
        bicc1f
        delay
        ...
1:      target
        bicc2f
        ...
```

- Do not put save/restore in the delay slot of an annulling BICC. If the save/restore is annulled, The microSPARC-IIep CPU must take a cycle to fix the current window pointer (CWP).

```
        bicc,a  1f
        save
```

- Do not follow multicycle instructions with a BICC.

```
        Does not Fold       Can Fold
        -------------       ---------------
        std                 std
        bicc                add
        delay               bicc
```

```
delay
```

- Do not follow WRspec with a BICC. Folding is disallowed when there is a WRspec anywhere in the pipeline's D, E, or W stages. WRspec refers to any of the special registers (PSR, WIM, TBR, Y).

```
Does not Fold        Can Fold
--------------       ---------------
mov .., %psr         mov .., %psr
nop                  nop
bicc                 nop
nop
nop
bicc
```

Note – Only integer branches are folded; FP branches are not. Calls are not folded owing to a register file limitation.

2.2.3 Multicycle Instructions

Most instructions in the microSPARC-IIep CPU take a single cycle to execute. The instructions listed in *Table 2-6* take multiple cycles.

Table 2-6 Instructions Taking Multiple Cycles

Instruction	Cycles
JMP, RETT	2
LDA, STA	2
LDD, LDDA, STD, STDA	2
LDSTB, LDSTBA	2
SWAP, SWAPA	2
STA FLUSH	3
IFLUSH	3
IMUL	19
IDIV	39

2.2.4 Pipeline Interlocks

The microSPARC-IIep CPU has several pipeline interlocks that may be avoided with improved code scheduling. The following operations result in interlocks.

- An integer load immediately followed by an instruction that uses the load destination as a source operand.
- A CALL followed by an instruction that uses r[15] of the register file as a source operand.
- A RD to a special register followed by a dependent operation.
- A folded SAVE/RESTORE that was annulled—it takes a cycle to fix the CWP.
- An unfolded CTI branch which is not taken and the delay slot + 1 instruction is not in the instruction queue.

2.2.5 Other Guidelines

Usage of the IMUL instruction, rather than a kernel routine, is preferred because of its higher performance. However, because it is highly dependent on the operand types, the performance of the IDIV instruction is not always better than that of a kernel routine.

2.2.6 Floating-Point Instructions

Scheduling of floating-point (FP) instructions can have a large impact on FP performance. The most important thing to consider when scheduling FP code is making efficient use of the floating-point queue. The FP unit has a three-entry floating-point queue and two independent functional units (multiplier and everything else).

The microSPARC-IIep CPU does not interlock for FP loads—including double-word loads—followed by dependent FP operations. Since operands for floating-point operations are read in W-stage, the result from the previous floating-point load can be bypassed to the floating-point units.

Refer to Section 4.6, *FP Performance Factors* for more information about floating-point performance.

2.2.6.1 FP Interlocks

- FP queue full - if an FPOP is in E-stage and the FP queue is full, the pipe must be held until the first instruction in the queue completes.
- FP store waiting for data from FPOP in queue - held in E-stage.

- FP load writing register used by FPOP in queue - held in W-stage. This applies whether the FPOP register is RS1, RS2, or RD.
- FPLD followed by FPST - single cycle interlock if the FP register (modulo 2) being loaded is the same as the FP register being stored (modulo 2).
- FPLDFSR/FPSTDFQ followed by any FPOP/FPMEMOP/FPCMP - single cycle interlock.
- FPOP followed by FPLDFSR/FPSTDFQ - LDFSR or STFSR must wait for FPOP to complete.
- FP branch in decode and FCCV (FP condition code valid) deasserted. The IU pipe will interlock until FCCV is reasserted. FCCV is deasserted when an FCMP is started and reasserted when the FCMP completes. The branch is held in D-stage.

2.2.6.2 Functional Units

There are two functional units: the multiplier and the Meiko core, which handles all other operations. The multiplier can start an operation every three cycles, but operations dependent on the multiplier results must wait five cycles for the result to be written. The initial multiply must also be in the first queue entry if the second multiply is to be started before the first results are written. The Meiko core is not pipelined; when an operation completes, the data and functional unit are both available. See Chapter 4, for details on instruction cycle count.

2.2.6.3 FP Queue Details

The FP queue is three entries deep. It allows out-of-order issue, but forces in-order completion. Only one operation can be started per cycle, and only one operation may complete per cycle. An operation does not leave the queue until it has written its results. The following examples demonstrate how dependencies affect the pipeline.

- Out-of-order issue, no dependencies - data written back in-order, issued out of order because of functional unit availability.

```
Unit        Issued                  Written
fmuld       %f0, %f2, %f4           Mult    x       x+5
fmuld       %f6, %f8, %f10          Mult    x+3     x+8
faddd       %f12, %f14, %f16        Adder   x+2     x+9
```

- FADD throughput - dependencies have no effect, 5 cycles per operation, due to a single functional unit.

```
Unit        Issued                  Written
faddd       %f0, %f2, %f2           Adder   x       x+5
faddd       %f2, %f2, %f0           Adder   x+5     x+10
```

- FMUL throughput - no dependency, 3 cycles per operation.

```
Unit        Issued                  Written
fmuld       %f0, %f2, %f4           Mult    x       x+5
fmuld       %f6, %f8, %f10          Mult    x+3     x+8
```

- FMUL throughput - with dependency, 5 cycles per operation.

```
Unit        Issued                  Written
fmuld       %f0, %f2, %f2           Mult    x       x+5
fmuld       %f0, %f2, %f2           Mult    x+5     x+10
```

- FMUL/FADD pair - no dependency, 5 cycles per pair. The second multiply cannot enter the queue until the first add has completed, at which time the second add is being started.

```
Unit        Issued                  Written
faddd       %f0, %f2, %f4           Adder   x       x+5
fmuld       %f6, %f8, %f10          Mult    x+1     x+6
faddd       %f0, %f2, %f4           Adder   x+5     x+10
fmuld       %f6, %f8, %f10          Mult    x+6     x+11
```

- FMUL/FADD pair - one dependency, 6 cycles per pair. It is immaterial which way the dependency goes.

```
Unit        Issued                  Written
faddd       %f0, %f2, %f4           Adder   x       x+5
fmuld       %f4, %f6, %f8           Mult    x+5     x+10
faddd       %f0, %f2, %f4           Adder   x+6     x+11
fmuld       %f4, %f6, %f8           Mult    x+11    x+16
```

```
Unit        Issued                  Written
fmuld       %f0, %f2, %f4           Mult    x       x+5
faddd       %f4, %f6, %f8           dder    x+5     x+11
fmuld       %f0, %f2, %f4           Mult    x+6     x+11
faddd       %f4, %f6, %f8           Adder   x+11    x+16
```

- FMUL/FADD/FMUL - two dependencies, 10 cycles per pair.

```
Unit        Issued                  Written
faddd       %f0, %f2, %f4           Adder   x       x+5
fmuld       %f4, %f6, %f0           Mult    x+5     x+10
faddd       %f0, %f2, %f4           Adder   x+10    x+15
fmuld       %f4, %f6, %f8           Mult    x+15    x+20
```

- Longer instructions (divide, square root) - other instructions can enter the pipeline, but none complete out of order. The integer pipe is not held unless a fourth FPop tries to enter the queue. Note that the second multiply cannot start until the first advances to the first queue entry.

```
Unit        Issued                  Written
fdivd       %f0, %f2, %f4           Adder   x       x+35
fmuld       %f6, %f8, %f10          Mult    x+1     x+36
fmuld       %f12, %f14, %f16        Mult    x+36    x+41
```

2.2.7 Loads and Stores

Load and store ordering has a large impact on the microSPARC-IIep CPU's performance. The microSPARC-IIep CPU has an 8-kilobyte write through, with write allocate, data cache. If all accesses hit the cache, the order of accesses makes little difference. The order of access can have a large effect on the latency of cache misses though. The following guidelines may help improve performance:

- Group memory accesses by DRAM page — Cache misses require reads from DRAM. The DRAM access is faster if it can be accessed in page mode. Therefore, loads and stores to the same page should be grouped together. One way to do this is to group accesses that use the same base register together, since these are likely to be in the same page. For instance:

  ```
  Poor order:                 Good order:
  ld      [%o0], %f3          ld  [%o0], %f3
  ld      [%i5-12], %f4       ld  [%o0+8], %f5
  ld      [%o0+8], %f5        ld  [%i5-12], %f4
  ld      [%i5-8], %f2        ld  [%i5-8], %f2
  ```

  See Section 2.3, *Using the Two Page-Hit Registers* for gaining further improvement by effectively using the page-hit registers.

- Minimize write buffer full penalty — the microSPARC-IIep CPU has four write buffers. At higher frequencies, the write buffers take more cycles to flush. So, use fewer store instructions, if possible, and reduce clustering of stores to allow the buffers a chance to empty. One technique is to maximize the use of store double (std). A double word store occupies only one write buffer entry and takes one memory access. Storing the two registers separately requires two write buffer entries and two memory accesses.

  Since the microSPARC-IIep CPU has four write buffers, up to four stores can be clustered together without stalling the pipe if the stores hit. However, all issued stores must be written to memory before the next cache miss can be processed. It is recommended that the number of instructions between the stores and the next memory access be roughly proportional to the number of stores, to allow time for the write buffer to empty.

- Minimize usage of STB and STH — Memory accesses have word write enables, so these instructions are implemented as a read-modify-write memory operation. This is slower than a normal store operation.

2.2.8 General Techniques

The following actions help performance, but are not microSPARC-IIep specific optimizations.

- Decrease instruction count
- Reduce integer load use interlock through better instruction scheduling.
- Reduce register window overflow / underflow
- Reduce cache miss rates, especially store miss rates

2.3 Using the Two Page-Hit Registers

The two page-hit resisters can be used to improve the microSPARC-IIep CPU's performance. See Section 5.7.1, *Processor Control Register* for information on how to enable page-mode operations.

Each time a virtual address (VA) is translated for a memory operation, the resulting physical address (PA) is compared to the PA of the previous memory operation stored in the page-hit registers. If the two PAs are within the same 4 kilobyte physical address space, the MMU signals that the current operation is a *page hit*. This indicates to the memory interface logic that there is no need to toggle the RAS lines to the memory, and the overall access is therefore much faster. Every memory access puts its page address into the page hit register.

In the microSPARC-IIep CPU, there are two page-hit registers. This allows the saving of two PAs for possible page hits. This also requires the memory interface to divide its memory into two groups, one for each page-hit register. Each group also has its own RAS lines. The actual banks of memory are set up so that every other bank / SIMM belongs to a given page-hit register.

The two page-hit registers especially help applications that alternate a large number of accesses between text and data. If there were only a single page-hit register, text and data accesses would thrash the page hit register and reduce its effectiveness.

With the existing page allocation software, the first bank's pages must be entirely used or allocated before any of the second bank's pages are used. This means that the benefit of having two page-hit registers is not seen until that point is reached. This also means that when an application is mapped, the banks are used serially, one after the other.

To maximize the benefit of the page-hit register scheme, the two registers have to be used as much as possible. This means dividing all of the available pages into two groups that correspond to the two page-hit registers. After this is done, a number of allocation preferences can be used. One group can be used for text and the other for

data, or simply alternate between the two groups. In the microSPARC-IIep CPU, the DRAM page size is 4 kilobytes, so alternating makes available a total of eight kilobytes of fast-access memory at any given time.

The difference between page and non-page access in the microSPARC-IIep CPU is four cycles for a page hit compared with 11 cycles for non-page hit.

Note – When the microSPARC-IIep CPU accesses main memory on behalf of the PCI Controller (PCIC) for PCI DMA accesses, the page-hit registers are marked invalid prior to the DMA access to prevent hitting into these registers incorrectly.

CHAPTER 3

Integer Unit

The microSPARC-IIep integer unit (IU) implements SPARC integer instructions as defined in the *SPARC Architecture Manual version 8*. This integer unit is derived from the integer unit of the microSPARC-II CPU. This implementation balances the needs of high-performance and low-cost while maintaining software compatibility. The only differences between the microSPARC-II and microSPARC-IIep integer units are noted in Section 3.13, *Compliance With SPARC Version 8* in this chapter.

3.1 Overview

The microSPARC-IIep integer unit is a CMOS implementation of the SPARC 32-bit RISC architecture version 8. Important features include:

- 5-stage instruction pipeline
- Branch folding
- Instruction and data cache streaming support
- Hardware implementation of IMUL and IDIV
- 136-register register file supporting eight register windows
- Interface to on-chip floating-point unit
- 4-deep instruction queue supporting instruction prefetching
- Little- and big-endian byte ordering support

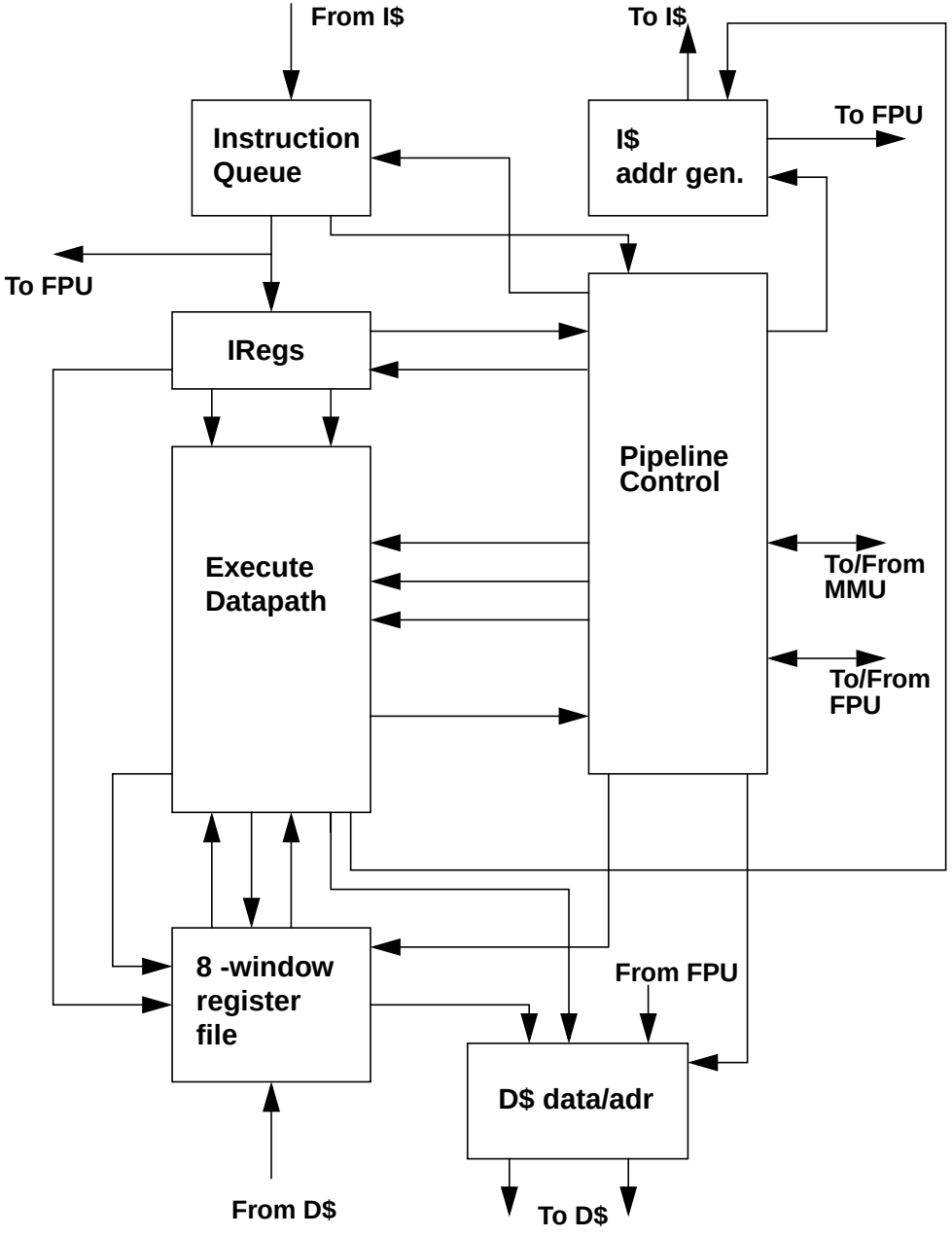


Figure 3-1 IU Block Diagram

3.2 Instruction Pipeline

The microSPARC-IIep IU uses a double (1-branch, 1-other) instruction issue pipeline with five stages.

1. F (Instruction Fetch): Instruction fetch occurs in this stage. Instructions may be fetched either from the 4-instruction deep queue or directly from the instruction cache. The instruction is valid at the end of this stage and is registered inside the IU.

2. D (Decode): This stage decodes the instruction and reads the necessary operands. Operands may come from the register file or from internal data bypasses. The register file has three independent read ports — two for operand or address calculation, and one for store operand read in the E-stage. For situations where the necessary operand is in the pipeline and has not been written to the register file yet, internal bypasses are supplied to prevent pipeline interlocks. In addition, addresses are computed for CALL and Branch in this stage.

3. E (Execute): This stage performs ALU, logical, and shift operations. For memory operations (e.g., LD) and for JMPL/RETT, the address is computed in this stage. Store operand read is done in this stage from the register file's third read port and sent to the data cache.

4. W (Write): This stage accesses the data cache. For cache reads, the data is valid by the end of this stage, at which point it is aligned as appropriate. Store data read out in the E-stage is written to the data cache at this time.

5. R (Result): This stage loads the result of any ALU, logical, shift, or cache read operation into the register file.

Table 3-1 lists the cycles per instruction.

Table 3-1 Cycles per Instruction

Instruction	Cycles
CALL	1
Single Loads	1
Jump/Rett	2
Double Loads	2
Single Stores	1
Double Stores	2
LDF/LDDF	1

Table 3-1 Cycles per Instruction *(Continued)*

Instruction	Cycles
STF / STDF	1
LDA / STA	2
LDDA / STDA	2
STA FLUSH	3
IFLUSH	3
Taken Trap	3
Atomic Load / Store	2

3.3 Memory Operations

3.3.1 Loads

All load operations take one cycle in the microSPARC-IIep IU except for LDD which takes two cycles. For LD, LDB, and LDH, the pipeline does the following:

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E — Address operands are added to compute the memory address. This address is presented to the cache in this stage.
3. W — Address is registered in the cache and access is started. Data is expected at the end of this stage. Any necessary alignment and sign extension is done by the data cache prior to being registered by the IU.
4. R — Data is registered in the IU and is written into the register file.

In the event of a cache miss, the miss signal is given to the IU in the W stage. The miss signal holds the pipeline. Once the miss data is available, the cache signals the IU to release the pipeline. The IU also registers the miss data into the appropriate R-stage register and writes it into the register file. As the cache line is being filled, the IU can accept additional data either from within the filling line or from another line that exists in the data cache.

An integer LDD takes 2 cycles to complete because of the use of 32-bit datapaths. The pipeline does the following operations:

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E — Address operands are added to compute the even memory address. This address is presented to the data cache in this stage.
3. W (E2) — The even memory address is registered in the cache and access is started. This data is sent to the IU. At the same time, the odd address is generated by the IU and sent to the cache.
4. R (W2) — The even word is registered in the IU and written to the register file. The odd word address is registered in the cache and its access is started.
5. R2 — The odd word is registered in the IU and written to the register file.

In the event of a cache miss, the miss signal is generated in the W-stage of the LDD. The miss holds the pipeline. When the cache receives the miss data, the IU control releases the pipeline, registers the even data into the R register, and writes it to the register file. It picks up the odd data in the next stage. As the cache line is being filled, the IU can accept additional data either from the filling line or from another line that exists in the data cache.

Floating-point load-single and load-double instructions (LDF/LDDF) operate like an integer load, except that the floating-point register file is loaded with the data coming from the data cache. In the case of LDDF, the instruction is executed in only one stage using the 64-bit datapath that exists between the data cache and FPU.

3.3.2 Stores

The microSPARC-IIep IU register file has three independent read ports. As a result, store operations take one cycle, except integer STD which takes two cycles. For integer stores and floating point single stores, the IU duplicates the store data on both words of the 64-bit bus from IU to data cache. For floating point store double, the words are aligned correctly.

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E — The store virtual address is computed in the ALU. The store operand is read from the third read port of the register file — this includes potential bypassing of results and a store aligner. If it is a floating point store of any size, operands are read from the floating-point file instead. Integer and floating point store data are correctly selected and sent to the data cache.
3. W — The store data is registered by the data cache and written.
4. R — The store is complete.

For integer STD the pipeline does the following:

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E (D2) — The address operands are added to compute the even memory address and sent to the data cache. This address is registered within the IU to provide the data cache with the odd address in the next stage. At the same time, the even store data is read from the register file's port 3 or bypassed from instructions still in the pipe and is sent to the data cache.
3. W (E2) — The odd address is sent to the data cache. The odd word is read from register file or bypassed from instructions still in the pipe and is sent to the data cache. The even word is written to data cache.
4. R (W2) — The odd word is written to the data cache.
5. R2 — The STD is complete.

3.3.3 Atomic Operations

SWAP and LDSTUB each take two cycles to complete. The pipeline does the following on the SWAP instruction:

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E (D2) — The address operands are added to compute the swap memory address. This address is sent to the data cache to start the cache read portion of the operation. The register to be swapped is read out in this stage and sent to the data cache.
3. W (E2) — The data cache returns the memory location accessed. The register to be swapped is sent to the data cache again. The store address is not sent to the data cache again.
4. R (W2) — The IU registers the read data and writes it to the register file.
5. R2 — The SWAP is complete.

The pipeline does the following on the LDSTUB instruction:

1. D — Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.
2. E (D2) — The address operands are added to compute the LDST address. This address is sent to the data cache to start the cache read portion of the operation. 0xffff.ffff is sent to the data cache along with the appropriate bytemarks for the store.

3. W (E2) — The data cache returns the memory location accessed and it is shifted appropriately and sent to the IU. 0xffff.ffff is sent to the data cache again. (The store address is not sent to the data cache again.)
4. R (W2) — The IU registers the read data and writes it to the register file.
5. R2 — LDSTUB is complete.

3.4 ALU / Shift Operations

Most ALU and shift operations take a single cycle to complete. The exceptions are integer multiply and integer divide. On add, subtract, boolean, and shift operations, the pipeline does the following operations:

1. D — Operands are read from the register file or bypassed from instructions still in the pipe
2. E — Appropriate operation is executed in ALU or shifter. There is a selective inverter on the B input of the ALU to allow for subtracts and certain Boolean operation (e.g. ANDN)
3. W — Result of operation is forwarded to the next stage
4. R — Result is stored in the register file

3.5 Integer Multiply

Integer multiply normally takes 22 cycles to complete, but may complete in 19 cycles if the three instructions preceding the multiply instruction do not write into the integer register file. The algorithm implemented in the microSPARC-IIep IU is a modified Booth's (2-bit) multiply. The multiply process can be broken up into four distinct steps:

1. Initialization: 1–4 cycles
2. Booth's iteration: 16 cycles
3. Correction (per Booth): 1 cycle
4. Writeback: 1 cycle

The first cycle is used to set up the registers used in the multiply. The RS1 and RS2 registers are initialized to the operands of the multiply. The W-stage result register and the RS2 register are used as accumulators. At the completion of the multiply, the W-stage register contains the most significant 32 bits of the result and the RS2 register contains the least significant 32 bits of the result. The W-stage register contents are then written to the Y register and the RS2 contents to the destination register in the register file.

3.6 Integer Divide

Integer divide normally takes 42 cycles to complete, but may complete in 39 cycles if the three instructions preceding the divide instruction do not write into the integer register file. If an overflow is detected, however, the instruction completes in six cycles. The algorithm implemented in the microSPARC-IIep IU is non-restoring binary division (add and shift). The divide process can be broken into five distinct steps:

1. Divide by zero detection: 1–4 cycles
2. Initialization / overflow detection: 3 cycles
3. Non-restoring division iteration: 33 cycles
4. Correction (for non-restoring): 1 cycle
5. Writeback: 1 cycle

Because the microSPARC-IIep IU does not allow traps to be taken in the middle of instructions, the first step is to determine if there is a divide by 0 condition.

The high order bits of the dividend are in the Y register. The low order bits are in the RS1 operand. The divisor is in the RS2 operand. In the initialization step, the Y register is read out and put into the RS1 register in the datapath. The RS1 operand is passed through to the W-stage register. The RS2 operand is passed to the RS2 register. The W-stage and RS1 registers are used as accumulators. At the completion of the divide, the W-stage register contains the final quotient.

There are two overflow options for signed divide with a negative result as defined in the SPARC version 8 manual. The microSPARC-IIep IU generates overflow when the result is less than -2^{31} with a remainder of zero.

If an overflow condition is detected, the divide terminates early with the appropriate result being written to the destination register.

If no overflow is detected, the non-restoring (sub and shift) divide stage is started. A correction step is provided to correct the quotient (necessary for this algorithm). After the correction step, the quotient is written to the correct destination register.

3.7 Control-Transfer Instructions

3.7.1 Branches

Branches are handled in two ways in the microSPARC-IIep CPU. A branch may be folded with its delay slot instruction or it may flow down the integer pipeline. Refer to Section 5.7.1, *Processor Control Register* for information on how to enable branch folding.

In order for a branch to be folded with its delay slot, several criteria must be met. Among these are:

- The branch, delay slot instruction, and the instruction following the delay slot must all be in the instruction queue or at the inputs of the IU from the instruction cache.
- No other control-transfer instruction (CTI) may be in the D-stage.
- No multi-cycle instruction may precede the branch.

A target instruction fetch is immediately started in the D-stage of the BICC / delay-slot pair. In addition, the delay slot + 1 instruction is sent to a special alternate buffer. All folded branches are predicted taken. In the next cycle, the target instruction may begin execution (if the delay slot is not a multi-cycle instruction). In this cycle, it may be determined that the branch was not taken, which results in the target instruction's being ignored and the delay slot +1 instruction's being fetched from the alternate buffer. Taken folded branches require zero cycles to execute, while untaken folded branches require one cycle to execute.

Nonfolded branches usually take a single cycle to execute. There is no penalty for taken compared with untaken branches, even if the instruction prior to the branch sets the condition codes, provided the delay slot + 1 instruction is in the instruction queue. In the event that the branch is untaken and the delay slot + 1 is not in the instruction queue, the branch takes two cycles.

In the Decode stage, the IU evaluates the condition codes and branch condition to determine whether the branch is taken or untaken. The IU outputs the correct instruction address for either the target or fall-through paths in time to be registered by the instruction cache for the fetch occurring in the next cycle. Refer to Section 2.2, *Compiler Optimization Guidelines* for more information.

3.7.2 JMPL

JMPL is a two cycle instruction in the microSPARC-IIep IU.

1. D — Read operands from register file or bypass from instructions still in the pipe. Sign extend immediate operands. The delay slot instruction is fetched in this stage.
2. E (D2) — Compute target address and send this to the instruction cache.
3. W(E2) — Fetch target.
4. R (W2) — Load the program counter (PC) of the JMPL instruction into the destination register.

3.7.3 RETT

RETT is a two cycle instruction in the microSPARC-IIep IU.

1. D — Read operands from register file or bypass from instruction still in the pipe. Sign extend immediate operands. The delay slot instruction is fetched in this stage.
2. E(D2) — Compute target address and send this to the instruction cache.
3. W(E2) — Fetch target.
4. R(W2) — Set PSR.ET to 1, move PSR.PS to PSR.S, and increment PSR.CWP.

3.7.4 CALL

CALL is a single cycle instruction in the microSPARC-IIep IU.

1. D — Add PC and disp30 to form target address. Send this address to instruction cache. The delay slot instruction is fetched in this stage.
2. E — The CALL target is fetched.
3. W — No action is taken
4. R — The program counter (PC) of the CALL is written to r[15].

3.8 Instruction Cache Interface

In the event of an instruction cache miss, the IU is informed of the miss early in the F-stage to prevent the pipeline from moving the missed instruction into the D-stage. The IU then waits for the instruction to be fetched. Once the missed instruction is returned, the IU releases the pipe and execution continues.

The instruction cache is implemented so that the missed word of the cache line is returned first. The IU is free to stream instructions from the instruction cache as the cache is doing its line fill. This means that the IU is not held for the entire duration of the cache fill, but it can use the instructions as soon as either the instruction cache receives it or, when fetching out of the filling line and that line is valid, directly out of the instruction cache. To do this, the IU is told when the instruction addressed by the IU is available to be registered. Then the IU either holds or releases the pipe.

If one of the instructions encountered during the instruction streaming is a taken CTI whose target is outside of the cache line being filled, and if that cache line is valid in the instruction cache, the fetch may take place. If the line is not in the cache, the IU holds and waits for that line to be filled after the previous line filling is completed.

3.9 Data Cache Interface

The data cache interface is roughly similar to the instruction cache interface. In the event of a data cache miss, the IU holds the pipeline in the W-stage.

The data cache is also implemented to return the missed word first. On Load instructions, when the data cache indicates that the load data is available, the data is passed through the load aligner for any necessary alignment. Afterwards the IU releases the pipe and strobes data into the R-stage (and the appropriate E-stage) register prior to its being written to the register file.

As for the instruction cache, the data cache can return data words as they are being filled. In addition, if, during a fill, a word is addressed from a different cache line, and if the line is valid in the data cache, that word is sent to the IU.

3.10 Interlocks

3.10.1 Load Interlock

There is a single-cycle load usage interlock in the microSPARC-IIep IU that is involved when a load instruction is followed by an instruction that uses the load operand (data) as a source operand.

3.10.2 Floating Point Interlocks

The IU interlocks the integer pipeline if it detects certain conditions in combinations of FP instructions. The single-cycle interlock is involved for the conditions:

- Floating-point load or load double in the E-stage of the pipe, a floating-point store or store double in the D-stage of the pipe and the FP register number (modulo 2) to be loaded is the same as the FP register number (modulo 2) to be stored
- A LDFSR or STDFQ operation in the E-stage of the pipe and the D-stage has any FP math operation, an FP compare, or any FP memory operation

In addition, the IU interlocks when the FPU deasserts the FCCV (floating point condition code valid) signal and the IU has a floating-point branch in D-stage. The IU continues to interlock the pipe until FCCV is reasserted. The FPU deassert FCCV when it begins an FCMP instruction and reasserts it when the FCMP is complete.

3.10.3 Miscellaneous Interlocks

Due to the datapath design, the microSPARC-IIep IU is unable to bypass special register read data to the instruction immediately following it in the pipeline. A single-cycle interlock occurs in those cases.

A CALL instruction followed by an instruction that reads R15 (destination register for the CALL), causes a one-cycle interlock.

IMUL and IDIV require datapath structures associated with the register file ports. As a result, they cannot use datapath bypass paths. If the three instructions preceding the IMUL or IDIV write the register file, the IU interlocks until these instructions have completed. The maximum length of this interlock is three cycles. The minimum

is zero. (Examples of instructions that do not write the integer register file are: stores, FPops, integer and floating point branches, and IFLUSH. NOP does write the register file, into Register 0.)

There are also interlocks associated with branch folding. These are dependent on queue, cache, and pipeline state.

3.11 Traps and Interrupts

3.11.1 Traps

The microSPARC-IIep IU implements all SPARC V8 traps except the optional traps:

- Data store error
- R-register access error
- Unimplemented FLUSH
- Watchpoint detected
- Coprocessor exception

Trap priorities are defined in SPARC version 8. If multiple traps occur during one instruction, only the highest priority trap is taken. Lower priority traps are ignored since it is assumed that lower priority traps will persist, recur, or are meaningless owing to the presence of the higher priority trap.

In the pipeline, the trap indication always occurs when the trapping instruction reaches the W-stage of the pipeline. Note that traps may be detected as early as the D-stage of the instruction, in which case the trap indication is piped to the W-stage of that instruction.

After the assertion of the TRAP signal, instructions following the trapped instruction in the pipeline and any instructions in the instruction queue are flushed out. The processor status register (PSR) is set with:

- Bit ET (enable trap) = 0
- Bit PS (previous status) = S (i.e., the state of the S bit at the time of the trap)
- Bit S = 1 (supervisor mode)
- Bits CWP = value of current window pointer at the time of the trap

Also field TT (trap type) of the (trap base register (TBR) is set to the corresponding trap code and the PC and nPC values at the time of the trap are written into r17 and r18. Instruction fetches then transfer operation to the trap vector as defined in the TBR.

The microSPARC-IIep IU does not allow traps during execution of multi-cycle instructions. There are no deferred integer traps. The IU detects and acts on deferred floating-point traps.

3.11.2 Interrupts

The microSPARC-IIep IU is interrupted using the PCI controller and the PCI interrupt request lines (IRL). The interrupt controller in the PCI controller selects the highest priority interrupting device. It then signals to the IU which is the highest priority interrupt on the IRL lines.

The interrupt levels for the PCI interrupt controller are programmable by software.

Two 32-bit timers in the PCI controller can be programmed to generate interrupts at any level required. Refer to Chapter 11, *Mode, Timing, and Test Controls*.

The PCI interrupt controller can be disabled and bypassed, which allows an external interrupt controller to generate the IRLs directly to the microSPARC-IIep IU.

To ignore glitches on the IRL lines, the IRL signals must be stable for at least two cycles. Only then does the IU initiate an interrupt request to the processor. This request is pipelined by one cycle. The interrupt is taken by the instruction currently in the W stage of the pipeline (or, if that instruction is a help instruction, by the next non-help W stage) if the IRL level is greater than the current processor interrupt level (PIL) and there are no higher priority traps that take precedence. A help instruction is a dummy instruction inserted whenever additional cycles are required to complete execution of certain instructions, for example, the second cycle on LDD. The help instruction propagates through the pipeline and maintains its integrity and consistency.

Note – Due to the one cycle delay existing between them when the IRL and PIL are compared and when the trap priorities are checked, a problem could arise where back-to-back PSR writes cause an interrupt to occur when the existing value in PSR.PIL is greater than the IRL. The microSPARC-IIep IU employs hardware to prevent this situation from occurring.

3.11.3 Reset Trap

On reset, the following steps occur:

- Traps are disabled (that is., PSR.ET ≤ 0) and supervisor mode is entered (that is, PSR.S ≤ 1)

- If the reset occurs during power-up, then PSR.PS, PSR.CWP, TBR.TT, r[17], and r[18] are undefined
- Otherwise, PSR.PS, PSR.CWP, TBR.TT, r[17], and r[18] are unchanged

- Execution begins at location PC=0 and nPC=4. Refer to section 9.9 for more information on programmable reset generation and section 10 for the reset controller operation

For more information, refer to Section 9.9, *System Status and System Control* on programmable reset generation and Chapter 11, *Mode, Timing, and Test Controls* on the reset controller.

3.11.4 Error Mode

Error mode is entered when a trap occurs and PSR.ET = 0. Entry into error mode causes the following occurrences.

- PSR.S ≤ 1; PSR.PS and PSR.CWP remain unchanged
- The contents of PC and nPC are stored into r[17] and r[18]
- PC and nPC are set to 0 and 4 respectively and the IU_ERROR signal is asserted

In addition, the TBR.TT may be changed if the trapping instruction is a RETT. The TBR.TT reflects:

- Privileged instruction trap when PSR.S = 0
- Underflow trap when a window underflow occurred
- Misaligned trap when a misaligned target address occurred

The IU remains in error mode until it is reset. For more information, refer to Section 9.9, *System Status and System Control* on programmable reset generation and Chapter 11, *Mode, Timing, and Test Controls* on the reset controller.

3.12 Floating-Point Interface

The microSPARC-IIep IU controls the addresses for all instructions and floating-point memory operations. The IU supplies the fetched instruction directly to the FPU from the instruction queue. The IU also informs the FPU if the instruction just loaded into the instruction register is valid.

For floating-point loads, the IU starts the cache access and the FPU reads the data. If the FPLOAD causes a data-cache miss, the IU sequences the cache miss. The FPU picks up the missed data once the IU releases the pipeline. For floating-point stores, the IU starts the cache access and picks up the store data from the FPU. The IU forwards this data to the data cache store bus.

The IU detects FP resource conflicts and interlocks the pipeline. In addition, the FPU may assert FHOLD to hold the IU pipeline when it detects an internal resource conflict. It deasserts FHOLD once the conflict is resolved.

FCC and FCCV are used by the IU to determine taken and untaken options for floating-point branches. If a floating-point branch is detected in decode stage and FCCV is not asserted, the IU stalls the pipeline until FCCV is asserted.

The FPU asserts the FEXC line when it detects a floating-point exception. The IU acknowledges the floating-point exception (FXACK) when the floating-point instruction is in the W-stage of the pipe. Then the IU takes a floating-point exception trap.

Floating-point operations take one cycle in the IU plus additional cycles in the FPU. For the number of cycles in the FPU, please refer to Chapter 4.

3.13 Compliance With SPARC Version 8

The microSPARC-IIep IU is designed to comply with the SPARC version 8 architecture, including hardware integer multiply and divide. However, it deviates from full support of SPARC version 8 features in the following ways:

- The microSPARC-IIep CPU has two additional bits in the PSR register for endian control. See Section 1.3.1, *Processor-internal Endian Support* for more information.
- Instead of decoding the eight bits of Address Space Identifier (ASI) for alternate space memory operations, the microSPARC-IIep MMU only decodes six bits and ignores the remaining two most-significant bits. Therefore, out-of-bound ASI encodings are not detected.
- The microSPARC-IIep IU does not implement the STBAR instruction since there is no need to force store ordering in this system. STBAR is interpreted as a read Y register operation with a null destination (%g0).
- The microSPARC-IIep IU does not support reads and writes to the ancillary state registers. All reads act like read Y register operations. All writes act like NOPs.
- When entering error mode, the microSPARC-IIep IU decrements the current window pointer (CWP) and updates R17 and R18. While not in conflict with the SPARC version 8 specification, this behavior is noted here.

- The hexadecimal value read from the implementation field (IMPL) of the microSPARC-IIep processor state register (PSR) is 0x0. The value read from the version (VER) field of the PSR is 0x4.

CHAPTER 4

Floating-Point Unit

The microSPARC-IIep floating-point unit (FPU) serves multiple purposes It executes floating-point instructions, detects data dependencies among those instructions, and handles floating-point related exceptions.

The FPU consists of a fast multiply unit, the Meiko core, and state machines to control the two datapaths. The Meiko core is licensed from Meiko, Inc.

Note – The floating-point unit of the microSPARC-IIep CPU is identical to that of the microSPARC-II, which has been in production and has gone through extensive laboratory and field testing.

This chapter covers the inner workings of the floating-point unit. For information relating to floating-point performance, refer to Section 2.2.6, *Floating-Point Instructions*.

4.1 Overview

The microSPARC-IIep floating-point unit (FPU) consists of the Meiko floating-point core and a fast multiplier.

The Meiko floating-point design implements the following algorithms which result in an optimized implementation of the floating-point engine.

- 8-bit multiply
- 2-bit division
- 1-bit square root
- Short distance (0-15 bits) shifter / normalizer
- Separate single-cycle rounding
- microcode state machine to control FPP and decode operation

The fast multiplier implements FMULS, FMULD, and FSMULD. In most cases, these operations can be executed in parallel while the Meiko core executes other floating-point instructions such as FADD. This ability to execute floating-point instructions in parallel provides significant instruction throughput.

In certain corner cases, the fast multiplier may not be able to complete multiplications. In such cases, the operation is aborted and restarted in the Meiko core instead. Nonetheless, the correct sequence of execution is maintained.

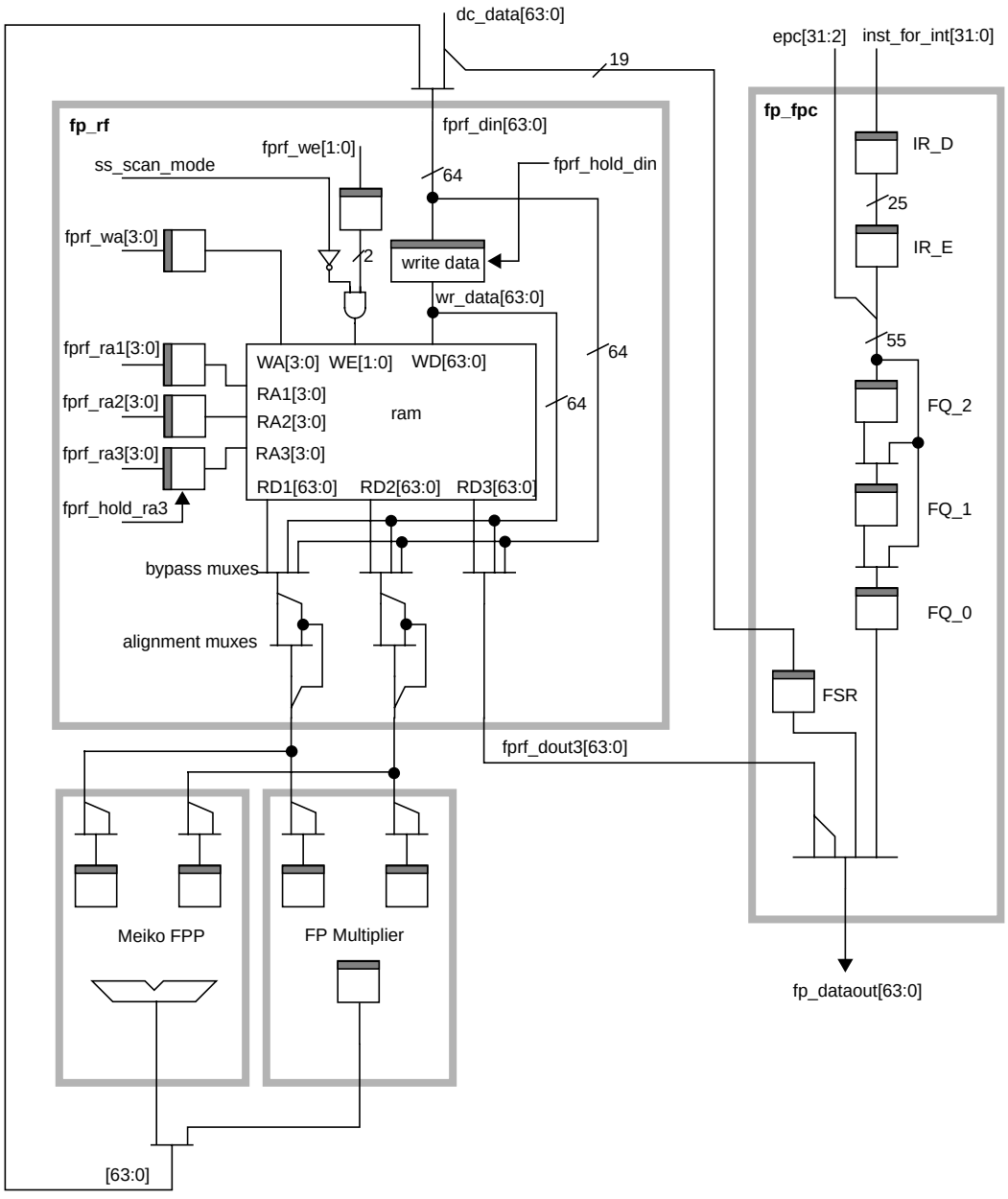


Figure 4-1 FPU Block Diagram

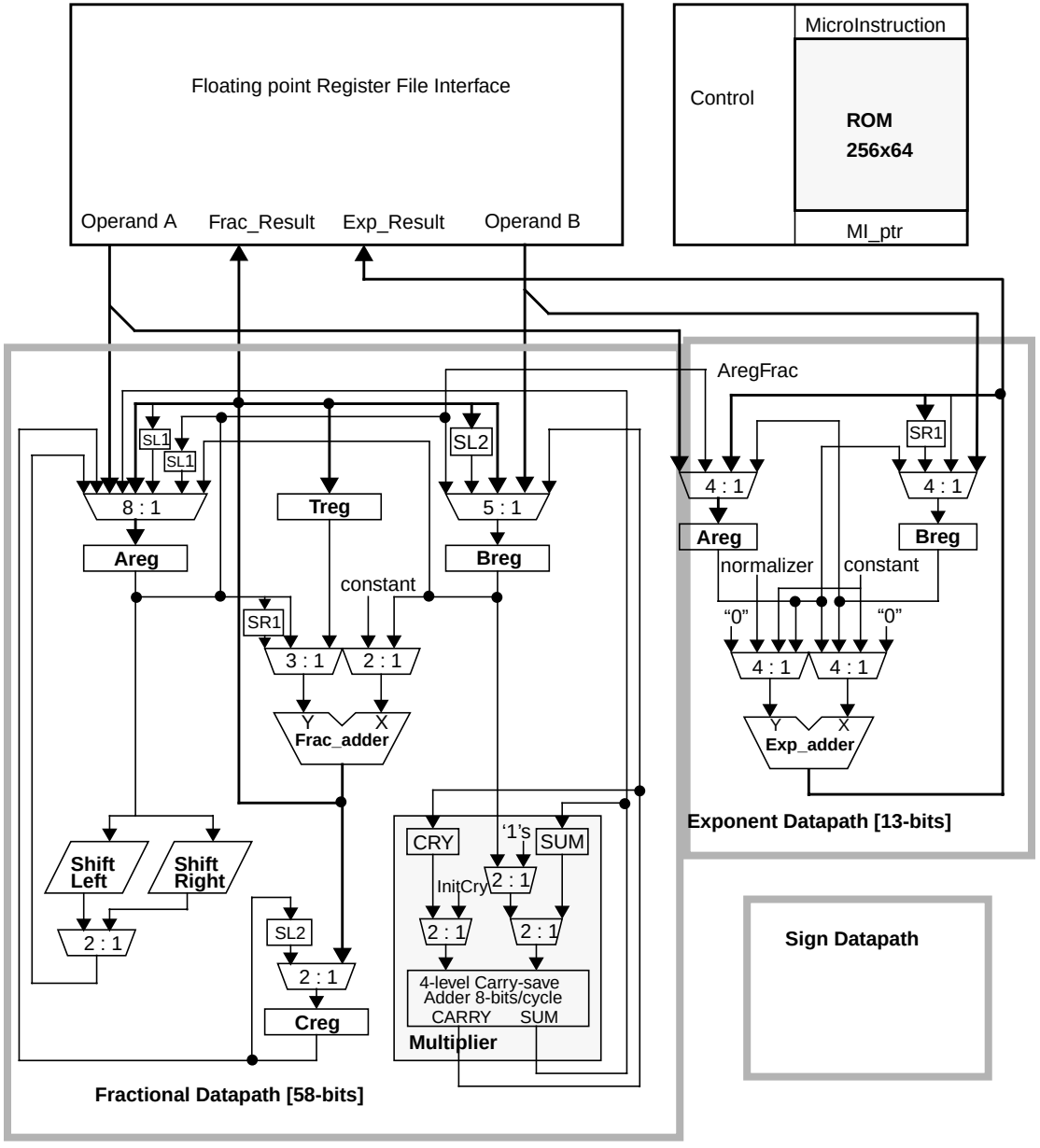


Figure 4-2 Meiko FPP Block Diagram

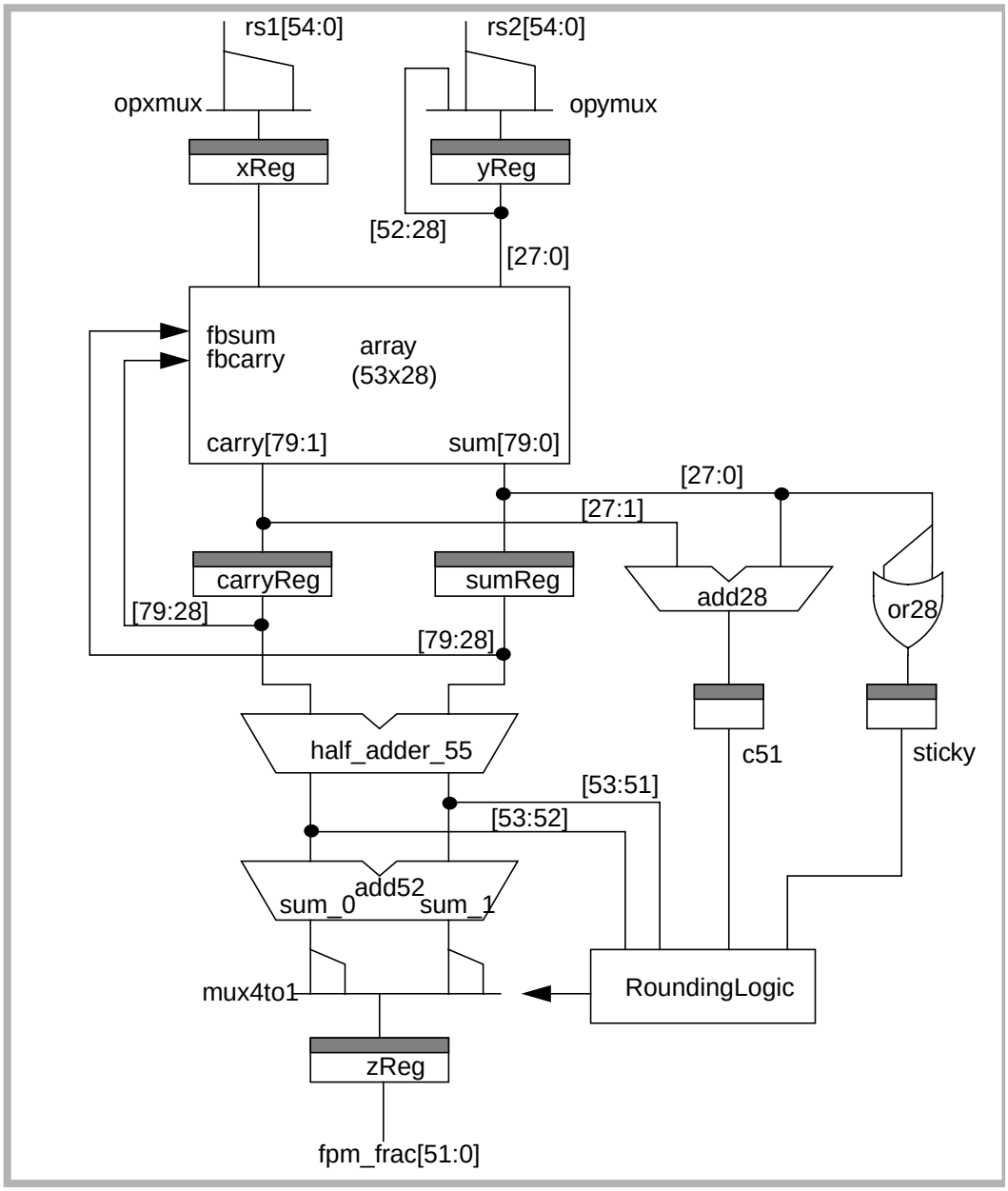


Figure 4-3 microSPARC-IIep Multiplier Mantissa Block Diagram

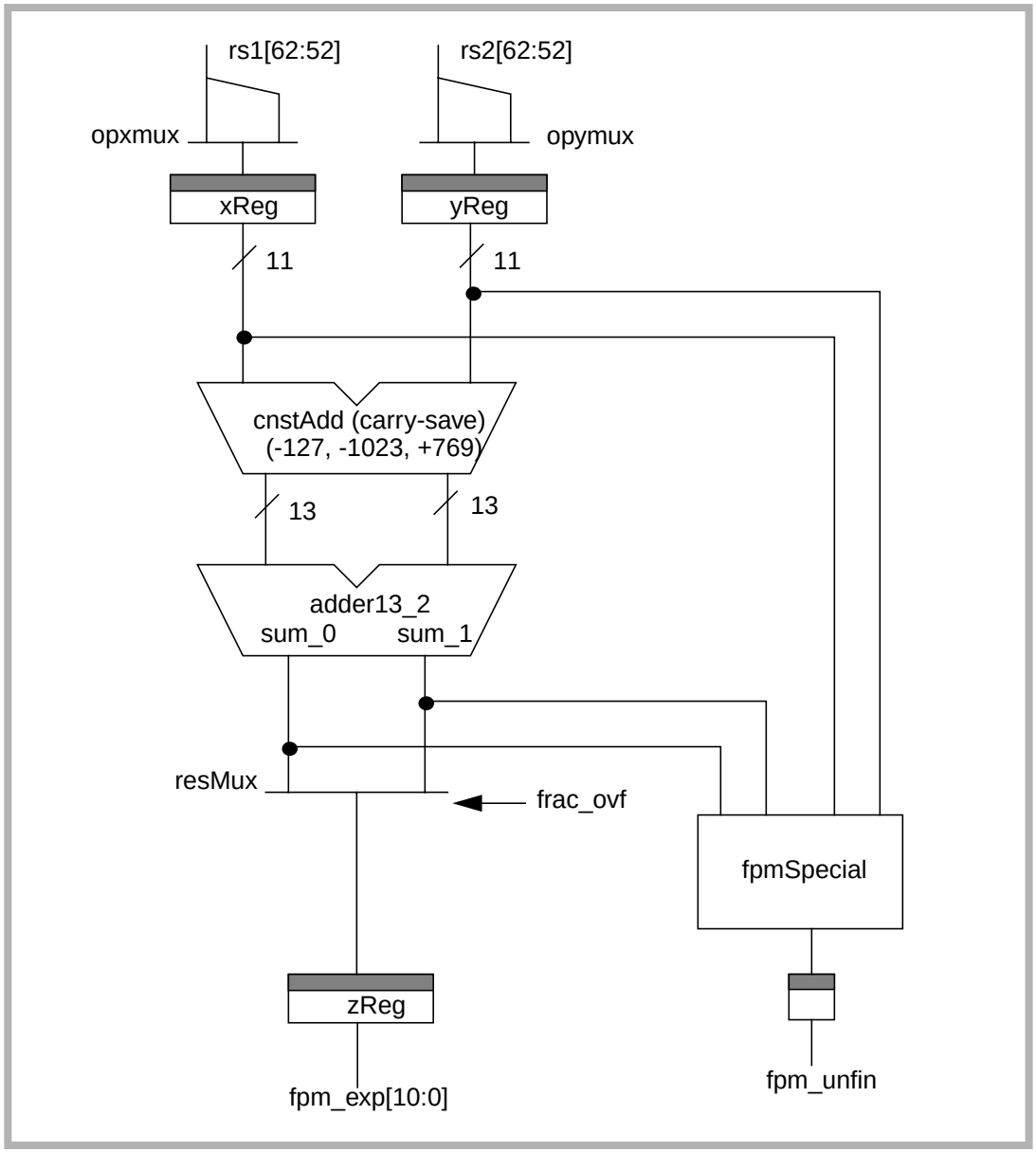


Figure 4-4 microSPARC-IIep Multiplier Exponent Block Diagram

4.2 FPU Internal Information

The FPC logic is partitioned into four main sections:

- FHOLD generation which occurs is cases of:
  - a full FQ with a FPop in pipeline (held in E-stage)
  - a dependent fp store (held in E-stage)
  - a dependent fp load (held in W-stage)
- FQ load control: An FPop can be loaded into the FQ if:
  - an entry is available, and
  - the FPU is in fp_execute mode
- FQ issue control: An FPop can be issued to the FPP if:
  - the FPop is in the FQ or E-stage of the pipeline and
  - the FPP is not busy, and
  - there are no data dependencies, and
  - the FPU is in fp_execute mode
- FQ writeback control: An FPop can write back its result if:
  - the FPop is in the front entry of the FQ, and
  - the FPP has finished execution and
  - the FPU is in fp_execute mode

The figures below show internal state diagrams and waveforms for some control signals.

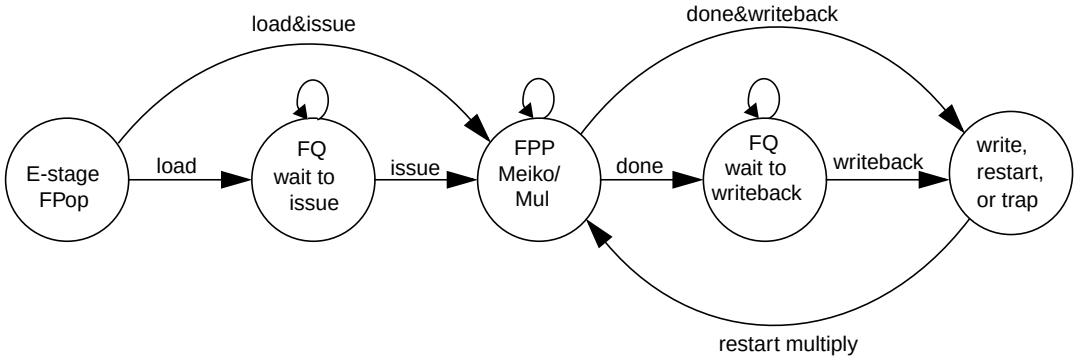


Figure 4-5 FPU Internal Control Flow Diagram

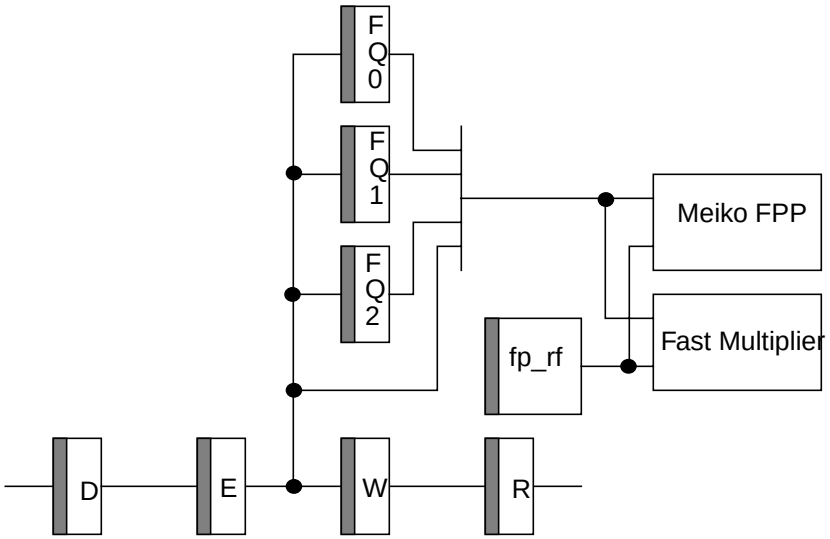


Figure 4-6 FPU Instruction Pipeline Diagram

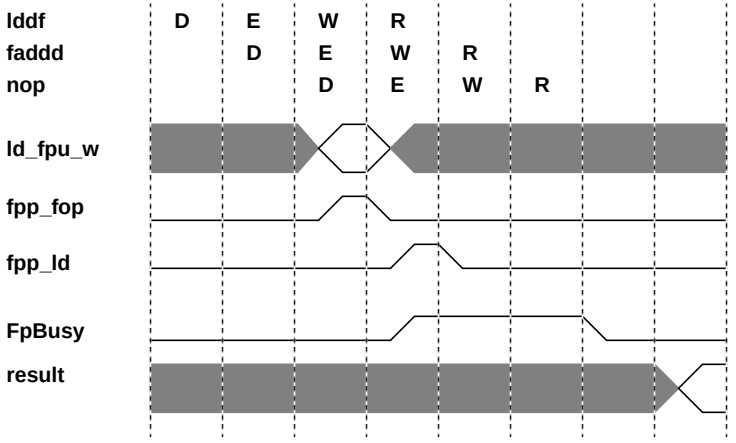


Figure 4-7 FPC / Meiko FPP Interface Waveforms

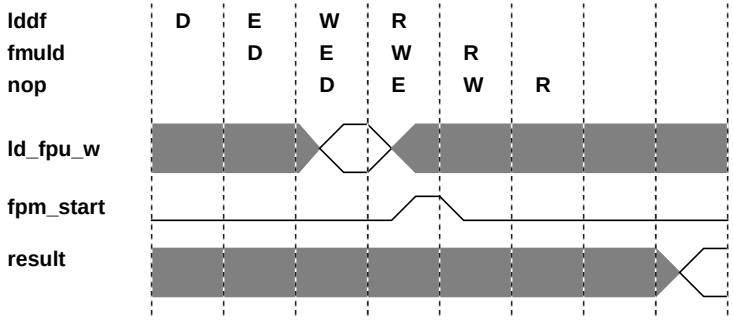


Figure 4-8 FPC / Multiplier FPP Interface Waveforms

4.3 Deviations from SPARC version 8

The microSPARC-IIep FPU supports all single- and double-precision floating-point (FP) instructions as defined in the SPARC Architecture version 8. Quad-precision floating-point instructions are not supported and execution of these instructions results in assertion of an unimplemented trap in the floating-point trap type (FTT) of the FSR. All implemented instructions except FSMULD complete in hardware. Therefore, the unfinished exception can only be generated by the execution of FSMULD.

The microSPARC-IIep floating-point unit also differs from the *SPARC IEEE 754 Implementation Recommendations* defined in Appendix N of the *SPARC version 8 Architecture Manual* in the NaN format. The following figures show the value returned for an untrapped floating-point result which is in the same format as the operands.

In *Figure 4-9*, all QNaN results have their sign bit set to zero.

		rs2 operand		
		number	**QNaN2**	**SNaN2**
rs1 operand	**none**	IEEE 754	QNaN2	ME_NaN
	number	IEEE 754	QNaN2	ME_NaN
	QNaN1	QNaN1	QNaN1	ME_NaN
	SNaN1	ME_NaN	ME_NaN	ME_NaN

ME_NaN: 0x7fff.0000 (single-precision)
0x7fff.e000.0000.0000 (double-precision)

Figure 4-9 Untrapped FP Result in Same Format as Operands

In *Figure 4-10*, QNaN2 is a copy of the mantissa bits of the operand with the extra low order bits zeroed and the sign bit zeroed.

	operand (rs2)			
operation	**+QNaN**	**-QNaN**	**+SNaN**	**-SNaN**
fstoi	ME_NaN	-imax	+imax	-imax
fstod	(QNaN2)	(QNaN2)	ME_NaN	ME_NaN
fdtos	ME_NaN	ME_NaN	ME_NaN	ME_NaN
fdtoi	ME_NaN	-imax	+imax	-imax

+imax = 0x7fff.ffff
-imax = 0x8000.000

Figure 4-10 Untrapped FP Result in Different Format

4.4 Implementation Specific Features

The microSPARC-IIep FPU implements a 3-entry floating-point deferred trap queue. When a floating-point instruction generates an *fp_exception*, the microSPARC-IIep CPU delays the handling of the *fp_exception* trap until the next floating-point instruction is encountered in the instruction stream. This implementation can be modeled as a state machine having three states: *fp_execute*, *fp_exception_pending*, and *fp_exception*. (see *Figure 4-11*.)

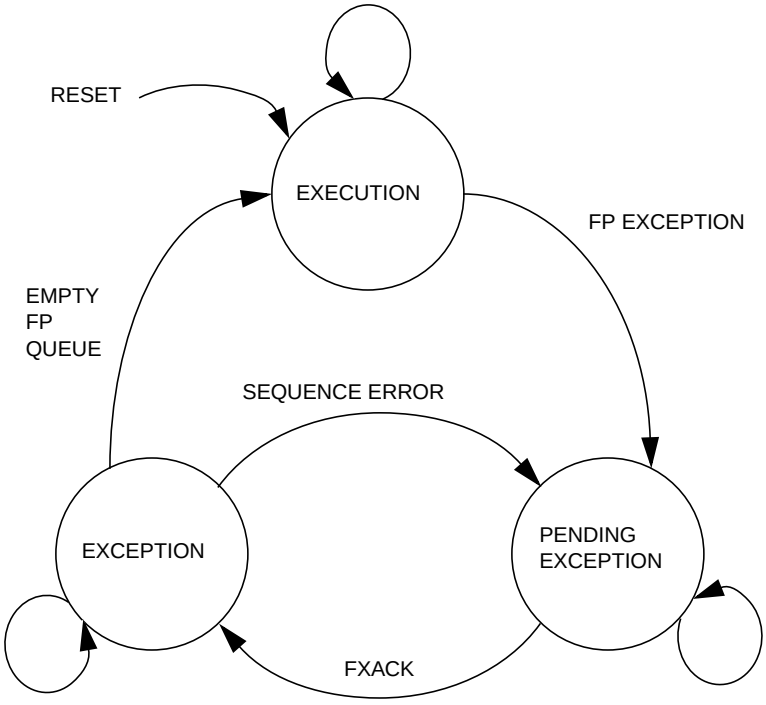


Figure 4-11 FPU Operation Modes

4.4.1 fp_execute State

Normally, the FPU is in *fp_execute* state. It transitions to *fp_exception_pending* when an floating-point operation results in a floating-point exception. If a STDFQ instruction is executed when the floating-point queue is empty, the FPU immediately generates an *fp_exception* trap while setting the Floating-point Trap Type (FTT) field of the Floating-point State Register (FSR) (bits 16 to 14) to sequence_error. However, in this case, the FPU remains in the *fp_execute* state.

4.4.2 fp_exception_pending State

The FPU moves from *fp_exception_pending* to *fp_exception* when the integer unit dispatches any floating-point instruction (including FBCC). The transition to *fp_exception* triggers a *fp_exception* trap. At this time, the first entry on the floating-point queue contains the instruction and address of the floating-point operation that caused the *fp_exception* originally. *fp_exception* traps can only be triggered when the FPU transitions from *fp_exception_pending* to *fp_exception*.

4.4.3 fp_exception State

While in the *fp_exception* state, the FPU can only execute floating-point store instructions such as STDFQ and STFSR. However, these instructions do not cause another *fp_exception* trap.

The FPU remains in the *fp_exception* state until the floating-point queue is emptied by STDFQ instructions. Once the queue is empty, the FPU returns to the *fp_execute* state. While in the *fp_exception* state, if the FPU encounters floating-point operations or floating-point load instructions, it returns to the *fp_exception_pending* state while setting the floating-point trap type (FTT) field in FSR (bits 16 to 14) to sequence_error, that is 0x4. However, the instruction triggering this sequence_error is not entered into the floating-point queue.

4.4.4 STDFQ Instruction

STDFQ stores the address and instruction from the floating-point queue to the effective address and effective address + 4 respectively.

4.5 Software Considerations

This section describes the software-visible features of the microSPARC-IIep floating-point unit.

The floating-point trap type (FTT) field is set whenever a floating-point operation completes or causes an exception. This field remains unchanged until another floating-point operation completes or causes a sequence_error. The FTT field can be cleared by executing a non-trapping floating-point operation such as fmovs %f0, %f0.

Table 4-1 describes the bits in the floating-point state register (FSR).

Table 4-1 Floating-Point State Register (FSR) Summary

Bits	Field	Description	Values	Writable by LDFSR
31:30	RD	Rounding Direction	0 — Round to nearest (tie even) 1 — Round to zero 2 — Round to +infinity 3 — Round to -infinity	Yes
29:28	res	reserved	Always 0	No
27:23	TEM	Trap Enable Mask	0 — Disables corresponding trap 1 — Enables corresponding trap	Yes
22	NS	Nonstandard FP	Always 0	No
21:20	res	reserved	Always 0	No
19:17	ver	FPU Version Number	Always 4	No
16:14	FTT	FP Trap Type	0 — None 1 — IEEE Exception 2 — Unfinished FPop 3 — Unimplemented FPop 4 — Sequence error	No
13	QNE	Queue Not Empty	0 — Queue empty 1 — Queue not empty	No
12	res	reserved	Always 0	No
11:10	FCC	FP Condition Codes	0 — == 1 — < 2 — > 3 — ? (unordered)	Yes
9:5	AEXC	Accrued Exception Bits	0 — No corresponding exception 1 — Corresponding exception	Yes
4:0	CEXC	Current Exception Bits	0 — No corresponding exception 1 — Corresponding exception	Yes

4.6 FP Performance Factors

The microSPARC-IIep FPU instruction cycle counts are provided in *Table 4-2*. The counts are in processor core clock cycles.

Table 4-2 FPU Instruction Cycle Counts

Instruction	Min	Typ	Max
FADDS	4	5	17
FADDD	4	5	17
FSUBS	4	5	17
FSUBD	4	5	17
FMULS	3	3	28
FMULD	3	3	35
FSMULD	3	3	3
FDIVS	6	20	38
FDIVD	6	35	56
FSQRTS	6	37	51
FSQRTD	6	65	80
FNEGS	2	2	2
FMOVS	2	2	2
FABSS	2	2	2
FSTOD	2	2	14
FDTOS	3	3	16
FITOS	5	6	13
FITOD	4	6	13
FSTOI	6	6	13
FDTOI	7	7	14
FCMPS	4	5	15
FCMPD	4	5	15
FCMPES	4	5	15
FCMPED	4	5	15
unimplemented	3	3	3

Because of the limited shifter size (0–15 bits was chosen to save hardware), the FP instruction cycle counts are data dependent. There are five ways in which operations may take longer than the typical cycle count:

- Exceptional operands (such as NaN, etc.) may add several cycles to the typical cycle count. In a normal environment, these are rare events probably caused by ill-conditioned data and are trapped (if traps are enabled).
- Possible exceptional results (results which are very close to underflow or overflow) may add up to five cycles to the typical cycle count. In a normal environment these are rare events, probably caused by ill-conditioned data.
- Denormalized operands add one extra cycle for each 15-bit shift required to normalize before the operation, and one extra cycle for each 15-bit shift required to denormalize the result after the operation (if necessary). Because operations on denormalized numbers always complete in hardware (except for the FSMULD instruction), the overall performance is greater than for an FPU which traps on denormalized operands.
- An add or subtract that requires an initial alignment of more than 15 bits adds one extra cycle for each 15-bit shift. Also, a subtract result that requires a shift of more than 15 bits to normalize adds one extra cycle for each 15-bit shift.
- Non-standard rounding modes (RZ and RN are the typical operating modes) may require up to three additional cycles for some corner cases and exceptions.

Statistical analysis shows that, on average, 90% of FPU instructions complete with the typical cycle count.

For a more detailed description of the Meiko FPP, please refer to the Meiko FPU specification, provided by Meiko Limited of Bristol, England.

The figures below show the peak performance (cached) of the microSPARC-IIep FPU for certain interesting FPOP combinations.

```
faddd  %f0, %f2, %f4
faddd  %f6, %f8, %f10
faddd  %f12, %f14, %f16
   .                                  5 cycles per instruction =
   .                                  20 MFLOPS @ 100 MHz
   .
faddd  %f0, %f2, %f4
```

Figure 4-12 FP Add Peak Performance

```
fmuld  %f0, %f2, %f4
fmuld  %f6, %f8, %f10
fmuld  %f12, %f14, %f16
   .                                  3 cycles per instruction =
   .                                  33.3 MFLOPS @ 100 MHz
   .
fmuld  %f0, %f2, %f4
```

Figure 4-13 FP Mul Peak Performance (No Dependencies)

```
fmuld  %f0, %f2, %f2
fmuld  %f0, %f2, %f2
fmuld  %f0, %f2, %f2
   .                                  5 cycles per instruction =
   .                                  20 MFLOPS @ 100 MHz
   .
fmuld  %f0, %f2, %f2
```

Figure 4-14 FP Mul Peak Performance (Dependency)

```
fmuld  %f0, %f30, %f0
faddd  %f10, %f12, %f12
fmuld  %f2, %f30, %f2
   .                                  5 cycles per instruction pair =
   .                                  40 MFLOPS @ 100 MHz
   .
fmuld  %f0, %f30, %f0
```

Figure 4-15 FP Mul-Add Peak Performance (No Dependencies)

```
fmuld  %f0, %f30, %f0
faddd  %f0, %f12, %f12
fmuld  %f2, %f30, %f2
   .                                  6 cycles per instruction pair =
   .                                  33.3 MFLOPS @ 100 MHz
   .
fmuld  %f0, %f30, %f0
```

Figure 4-16 FP Mul-Add Peak Performance (Dependency)

CHAPTER 5

Memory Management Unit

The microSPARC-IIep memory management unit (MMU) provides the functions specified in the SPARC version 8 Reference MMU Architecture. The implementation of the microSPARC-IIep MMU is based on the microSPARC-II MMU design. However, minor changes were made which include a separate dedicated I/O translation lookaside buffer (IOTLB) for translating I/O memory references. The IOTLB resides in the PCI controller and is separate from the CPU translation lookaside buffer (TLB).

Note – The changes to the microSPARC-II CPU that result in processor-visible differences for the microSPARC-IIep CPU are reflected in the MMU and MMU registers. This chapter details most of those changes. However, the addition of the endian control bits for the processor are defined in the processor state register (PSR) and described in Section 3.13, *Compliance With SPARC Version 8*.

5.1 Overview

The microSPARC-IIep MMU provides four primary functions:

- It translates the 32-bit virtual address of each running process to a 31-bit physical address. This translation is speeded up with the assistance of a 32-entry translation lookaside buffer (TLB). The MMU uses the three most significant bits of the physical address (PA[30:28]) to map to eight separate address spaces (see Appendix B, *Physical Memory Address Map*). It also supports 256 contexts.
- It provides memory protection to prevent unauthorized processes from reading or writing another process' address space.
- It implements virtual memory by maintaining page tables in main memory. When an address translation miss occurs, it performs a table-walk in the hardware and the resulting page-table entry is cached in the TLB.

- It arbitrates memory references among instruction and data caches, I/O, and TLB.

The microSPARC-IIep MMU contains a 32 entry fully-associative TLB and uses a pseudo-random algorithm for the replacement of TLB entries.

The address and data path block diagram of the microSPARC-IIep MMU is shown in *Figure 5-1*.

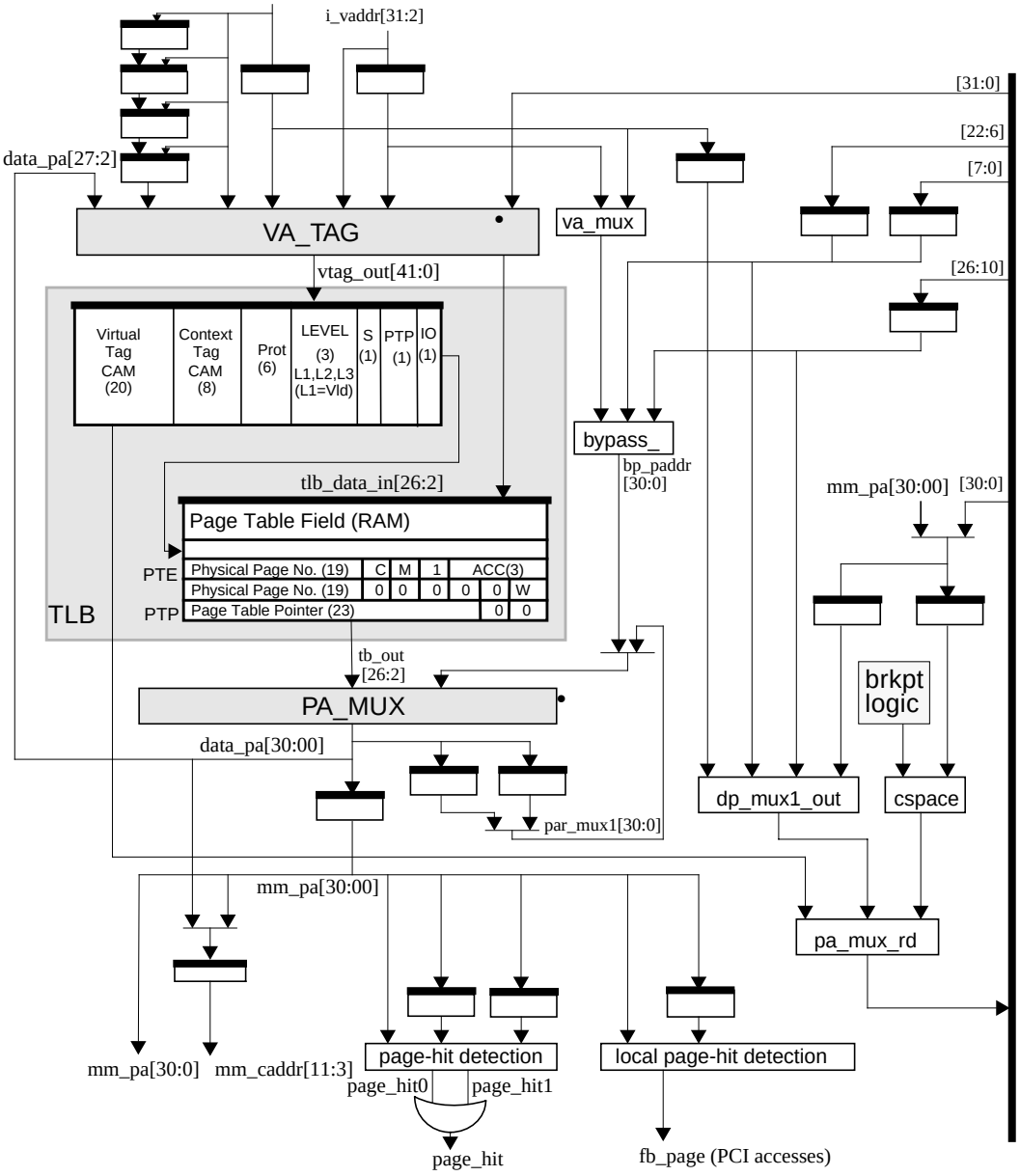


Figure 5-1 MMU Address and Data Path Block Diagram

5.2 MMU Programming Interface

The MMU internals are user-accessible using the load and store from alternate space instructions with the following address space identifiers (ASI).

- ASI = 0x03: Reference MMU flush or probe (see Section 5.6, *CPU TLB Flush and Probe Operations*)
- ASI = 0x04: Reference MMU registers (see Section 5.7, *Processor MMU Registers*)
- ASI = 0x06: Reference MMU diagnostics (see Section 5.14, *Diagnostic Features*)
- ASI = 0x20: Reference MMU bypass (see Section 5.12, *Translation Modes*)

5.3 Translation Lookaside Buffer

The TLB is a 32 entry, fully-associative cache of page descriptors. It caches CPU virtual-to-physical address translations and the associated page protection and usage information. The pseudo-random replacement algorithm determines which of the 32 entries should be replaced when needed. In the descriptions that follow the terms VA and PA are used to describe any virtual address (wb_vaddr, i_vaddr or d_vaddr) or physical address (mm_pa, or mm_caddr) respectively. This TLB is not used for IO translations, and only contains application translation information.

Note – The TLB operates in a fixed big-endian mode, therefore all entries should be stored using this mode.

5.3.1 TLB Replacement

The TLB uses a pseudo random replacement scheme. There is a six-bit counter in the TLB Replacement Control register (TRCR) which is incremented by one during each CPU clock cycle to address one of the TLB entries. When a TLB miss occurs, the counter value is used to address the TLB entry to be replaced. On reset the counter is initialized to zero. There is also a bit in the TRCR which is used to disable the counting function. See *Figure 5-2*. Additionally the TRCR has a programmable six-bit field which defines the counter “roll-over” point. This effectively locks down entries beyond the roll-over point, and prevents their replacement by subsequent translations. For the microSPARC-IIep CPU this roll-over point should be set to allow 32 entries at most, but can be set to a lesser value ensuring that some TLB entries are locked.

The microSPARC-IIep TLB supports another locking mechanism. By programming bits 16 to bits 14 of the TRCR, the first three TLB entries are exclusively reserved for page table pointers (PTP). In this case, PTPs can only be stored in those three TLB entries. This mechanism prevents PTPs from displacing page table entries (PTE) or vice versa.

Refer to Section 5.7.6, *TLB Replacement Control Register* for more information.

The MMU can store level 2 PTPs with a virtual tag or a physical tag. This is controlled using a bit in the TLB Replacement Control Register. When physical tags are enabled the MMU table walk algorithm starts with a root level access if a PTE were not found on the initial look-up. If, however, virtual tags are enabled for PTP2, the table walk algorithm searches for a virtually tagged PTP2 following the initial PTE miss. Should the virtually tagged PTP not be found, the root-level walk is started. When a virtually tagged PTP2 is found, the root, level 1 and level 2 look-ups can be bypassed, and a PTE can be immediately read from memory.

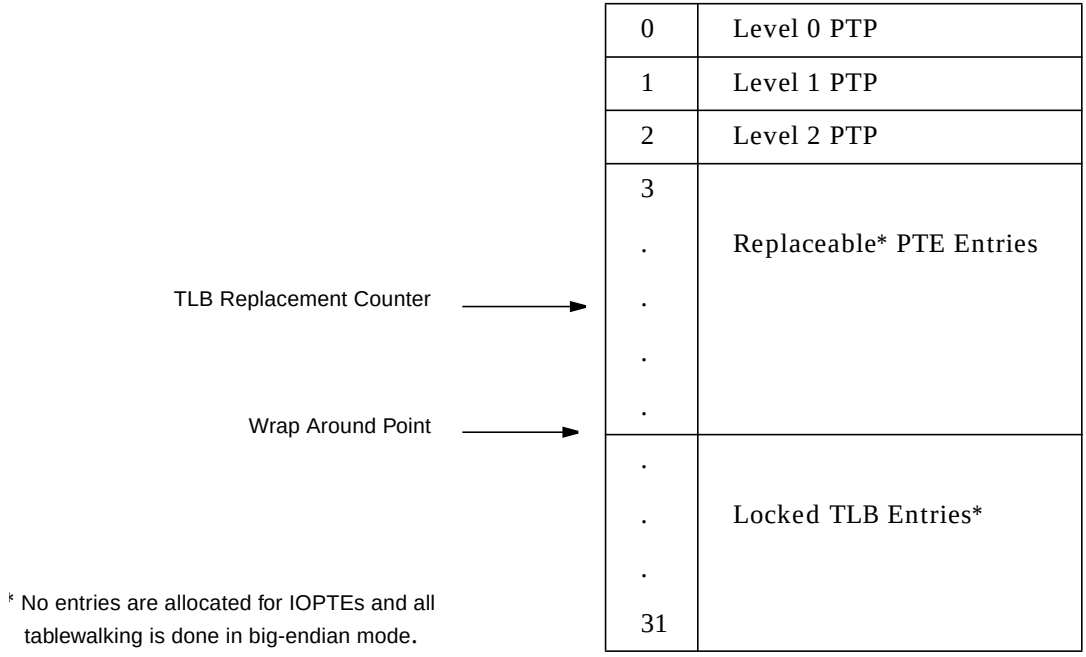


Figure 5-2 Possible TLB Replacement

5.3.2 TLB Entry

An entry in the TLB has the following fields: a virtual address tag, a context tag, a PTE level field, and a page table field.

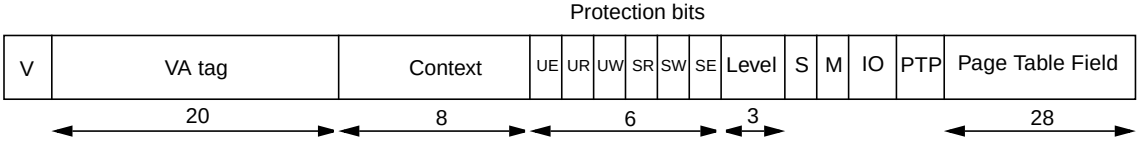


Figure 5-3 TLB Entry

Field Definitions:

- Valid (V) - This bit is used to indicate that the entry holds valid information.
- Virtual Address Tag —The 20 bit virtual address tag represents the most significant 20 bits (VA[31:12]) of the virtual address used to reference a PTE. VA[11:00] is the byte offset within a page. The address in this field is physical when referencing PTPs with the least significant bits containing PA[26:08].
- Context Tag — The 8-bit context tag comes from the value in the context register as written by memory management software when referencing PTEs. Both it and the virtual address tag must match the CXR and VA[31:12] to have a TLB hit. This field contains a physical address (PA[07:02]) when referencing PTPs. Note that for PTPs only six bits are used to hold PA[07:02], the two lower bits are always set to zero.
- Prot — The six protection bits in each TLB entry represent the decoded ACC bits from the matching PTE, namely user Rd, Wr, Ex, and supervisor Rd, Wr, Ex. These bits are used to check for protection violations on entries that meet the TLB hit criteria.
- Level — The 3-bit level field is used to enable the proper virtual tag match of region and segment PTE's. PTP's have this field set to use Index 1, 2 and 3 (b'000').

Table 5-1 Virtual Tag Match Criteria

Match Level	Match Criteria
111	None
011	Index1 (VA[31:24]
001	Index 1,2 (VA[31:18])
000	Index 1, 2, 3 (VA[31:12])

- Supervisor (S) — This bit is used to disable the matching of the context field indicating that a page has a supervisor level (ACC = 6 or 7).
- Modified (M) — This bit is set to a one when the page is written.
- IO Page Table Entry (IO) — This bit indicates that an IOPTE resides in this entry of the TLB. For the microSPARC-IIep, this bit is never set.
- Page Table Pointer (PTP) — This bit indicates that a PTP resides in this entry of the TLB. Note that all SRMMU flush types (except page) flush all PTPs from the TLB.
- Page Table Field — The page table field can either be a Page Table Entry (PTE), or an Page Table Pointer (PTP). This field can be read and written using ASI 0x06.

5.3.3 Page Table Entry

A Page Table Entry (PTE) defines both the physical address of a page and its access permissions. A PTE is defined for SPARC reference MMUs as follows.

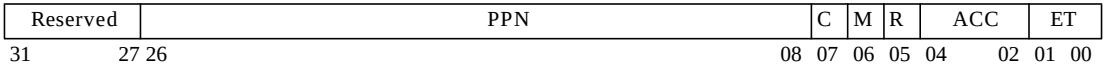


Figure 5-4 Page Table Entry in Page Table

Field definitions:

- Reserved (Rsvd) — Bits [31:27] should be written as zero, and are read as zero.
- Physical Page Number (PPN) - This field contains the high order 19 bits ([30:12]) of the 31-bit physical address of the page. The PPN appears on PA[30:12] when a translation completes.
- Cacheable (C) — When this bit is set to a one the page is cacheable by an instruction and/or data cache.
- Modified (M) — This bit is set to a one when the page is written to.
- Referenced (R) — This bit is set to a one when the page is accessed. All PTEs in the TLB have this bit set when the entry is loaded.
- Access Permissions (ACC) — These bits indicate whether access to this page is allowed for the transaction being attempted. The Address Space Identifier (ASI) determines whether a given access is a data access or an instruction access, and whether the access is being done by the user or supervisor. The field is defined as shown in *Table 5-2*.

Table 5-2 Page Table Access Permission

ACC	Access Mode	
	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

- Entry Type (ET) — This field differentiates the entry types in the TLB. Note that the entry type is not kept in the TLB RAM. On a probe operation the ET field is derived from a combination of other bits. See *Table 5-3* for the bit definitions of the ET field.

Table 5-3 Page Table Entry Types

ET	Entry Type
0	Invalid
1	Page Table Pointer
2	Page Table Entry
3	Reserved in Page Tables.

"Invalid" means that the corresponding range of virtual addresses is not currently mapped to a physical address.

Figure 5-5 shows the PTE format in the TLB RAM.

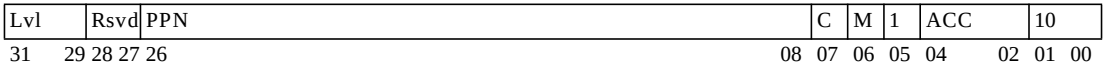


Figure 5-5 Page Table Entry in TLB

- Bits [28:27] are not implemented, should be written as zero, and are read as zero.
- Bit [05] is set to one by hardware indicating that every PTE in the TLB has been referenced.

- Bits [01:00] are set to 2'b10 by hardware indicating the entry type (ET) of a PTE. These bits are not actually stored in the TLB rather are derived as a function of the PTP bit of the tag.
- Bits[31:29] are set to indicate the page table level where the entry is to be found. The following table describes the possible encodings:

Table 5-4 Page Table Entry Level in TLB

LVL[2:0]	Page Table Level
000	Level 0 (Root)
100	Level 1
110	Level 2
111	Level 3

5.3.4 Page Table Pointer

A Page Table Pointer (PTP) contains the physical address of a page table and may be found in the Context Table, in a Level 1 Page Table, or in a Level 2 Page Table. Page Table Pointers are put into the TLB during tablewalks and removed from the TLB either by natural replacement (also during tablewalks) or by flushing the entire TLB. Note that the Level (Lvl) field in a PTP tag is always set to 0x7. *Figure 5-6* shows the PTP definition

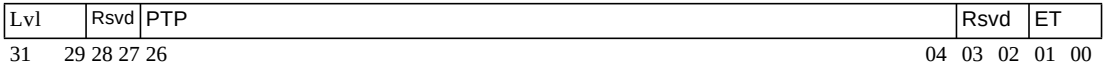


Figure 5-6 Page Table Pointer in Page Table

Field definitions:

- Reserved (Rsvd) — Bits[28:27, 03:02] should be written as zero, and are read as zero.
- Page Table Pointer (PTP) — The physical address of the base of a next level page table. The PTP appears on PA[30:08] during miss processing. The page table pointed to by a PTP must be aligned on a boundary equal to the size of the page table. Note that this is also true of the context table at the root level. The sizes of the tables are summarized in *Table 5-5*.

Table 5-5 Size of Page Tables

Level	Size (Bytes)
Root	1024
1	1024
2	256
3	256

- Entry Type (ET) — This field differentiates the entry types in the TLB. Note that the entry type is not kept in the TLB RAM. On a probe operation the ET field is derived from a combination of other bits. *Table 5-6* gives its bit definitions.

Table 5-6 Page Table Entry Types

ET	Entry Type
0	Invalid
1	Page Table Pointer
2	Page Table Entry
3	Reserved

"Invalid" means that the corresponding range of virtual addresses is not currently mapped to a physical address.

In the TLB, a PTP has the format of *Figure 5-7*.

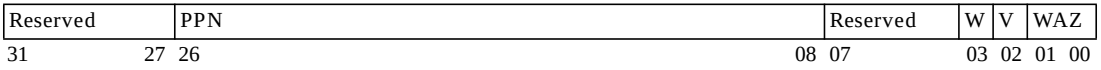


Figure 5-7 Page Table Pointer in TLB

- Bits [28:27] are not implemented, should be written as zero, and are read as zero.
- Level (Lvl) — The level bits for a PTP indicate the level at which the PTP is found. *Table 5-7* shows the possible level encodings.

Table 5-7 Page Table Entry Types

Lvl[2:0]	PTP level
000	Level-0 (root)
100	Level-1
110	Level-2

- Bits [03:02] are set to zero by hardware and are unused.
- Bits [01:00] are set to 2'b01 by hardware indicating the entry type (ET) of a PTP. These bits are not actually stored in the TLB; rather they are derived as a function of the PTP bit of the tag.

5.4 Address Space Decodes

The physical address space for microSPARC-IIep is decoded into eight address spaces, based on the upper three bits of the physical address(pa[30:28]). *Table 5-8* defines the address spaces and their decodes:

Table 5-8 Virtual Tag Match Criteria

PA[30:28]	Address Space
000	Main Memory Space
001	Control Space (Sun-4M system registers)
010	Flash ROM Space
011	PCI Space
100	Reserved I/O Space: Should not be accessed.
101	Reserved I/O Space: Should not be accessed.
110	Reserved I/O Space: Should not be accessed.
111	Reserved I/O Space: Should not be accessed.

5.5 CPU TLB Lookup

A virtual address to be translated by the MMU is compared to each entry in the TLB. During the TLB lookup the value of the Level field specifies which index fields are required to match the TLB virtual tag as shown in *Table 5-9*.

Table 5-9 Virtual Tag Match Criteria

Level	Match Criteria
111	None
011	Index1 (VA[31:24]
001	Index 1,2 (VA[31:18])
000	Index 1, 2, 3 (VA[31:12])

In addition to the virtual tag match, context matching of a PTE is required for all user page references (ACC is 0 to 5) when made by either user or supervisor (ASI = 0x8–0xB). Context matching is not required for a supervisor page reference (ACC is 6 or 7) when made by a supervisor (ASI = 0x9 or 0xB). This case takes advantage of the Supervisor bit in the TLB tag. Note that user references (ASI = 0x8 or 0xA) to supervisor pages (ACC is 6 or 7) result in address exceptions.

Note that the TLB ignores access level checking during probe operations. The most significant Level field bit is used as a Valid bit for the TLB. This means that root level PTEs are not supported.

5.6 CPU TLB Flush and Probe Operations

The flush operation allows software invalidation of TLB entries. TLB entries are flushed by using a store alternate instruction. The probe operation allows testing the TLB and page tables for a PTE corresponding to a virtual address. TLB entries are probed by using a load alternate instruction. The ASI value 0x3 is used to invalidate or probe entries in the TLB. In an alternate address space used for probing and flushing the address is composed as follows:

Figure 5-8 CPU TLB Flush or Probe Address Format

Field Definitions:

- Virtual Flush or Probe Address (VFPA) — This field contains the address that is used as the match criterion for the flush or probe operations into the TLB. Depending on the type of flush or probe, not all 20 bits are significant. Note that context flush uses the current context ID as defined in the context register.
- Type — This field specifies the extent of the flush or the level of the entry probed.

- Reserved - These bits are ignored. They should be set to zero.

5.6.1 CPU TLB Flush

The flush operation must remove the PTEs and PTPs from the TLB that match the type criteria in *Table 5-10*.

Table 5-10 TLB Entry Flushing

Type	Flush	PTE Match Criteria
0	Page	((ACC ≥ 6) OR CID match) AND VA[31:12] match
1	Segment	((ACC ≥ 6) OR CID match) AND VA[31:18] match
2	Region	((ACC ≥ 6) OR CID match) AND VA[31:24] match
3	Context	(ACC ≥ 6) OR CID match
4	Entire	None (Entire TLB Flush)
5 to F	Reserved	–

Page flush only removes matching PTEs from the TLB. All of the flushes remove matching PTEs and all PTPs from the TLB. CPU flush context operations flushes PTEs that match the current context, and all PTEs that have the S (Supervisor) bit set in their tags. If the CPU is running with virtual PTPs enabled, all virtually tagged PTPs are flushed for any occurrence of flush context, region, or segment. Flush operations to types 5-F are reserved and do not affect the TLB

5.6.2 CPU TLB Probe

The probe operation returns either a PTE from a page table in main memory or the TLB or a zero if there is an invalid address or translation error while searching for the entry implied by the probe. If there is an error, a zero is returned for data. The reserved probe types (0x5–0xF) return an undefined value. A type-4 probe (entire) brings the accessed PTE and any PTPs that are needed into the TLB. If the PTE were not already there, the referenced bit is updated. Probe types 0–3 affect one entry of the TLB which is invalidated at the end of the probe operation.

The value returned by a probe operation is specified in *Table 5-11*.

Table 5-11 Return Value for MMU Probes

Type	If No Memory Errors Occur																Mem Err
	Level - 0 Entry Type[1, 2]				Level - 1 Entry Type				Level - 2 Entry Type				Level - 3 Entry Type				
	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	pte	res	inv	ptp	
0(page)	0	0	0	→	0	0	0	→	0	0	0	→	X	0	X	0	0
1(seg)	0	0	0	→	0	0	0	→	X	0	0	X	—	—	—	—	0
2(reg)	0	0	0	→	X	0	X	X	—	—	—	—	—	—	—	—	0
3(ctx)	X	0	X	X	—	—	—	—	—	—	—	—	—	—	—	—	0
4(entire)	X	0	0	→	X	0	0	→	X	0	0	→	X	0	0	0	0
5-0xF	(undefined)																

1. pte = page table entry
   res = reserved
   inv = invalid
   ptp = page table pointer
2. For a given probe type, the table is read left-to-right:
   "0" = a zero is returned
   "X" = the page table entry itself is returned
   "→" = the next-level page table entry is examined
   "—" = don't care

5.7 Processor MMU Registers

The Processor Control register (CR) contains general CPU control and status flags. The current context identifier is stored in the Context Register (CXR), and a pointer to the base of the context table in memory is stored in the Context Table Pointer Register (CTPR). If an MMU fault occurs on a CPU-initiated transaction the address causing the fault is placed in the Synchronous Fault Address Register (SFAR) and the cause of the fault can be determined from the contents of the Synchronous Fault Status Register (SFSR). The TLB Replacement Control Register is used to control which TLB entries are to be replaced next. All of these internal MMU registers can be accessed directly by the processor through alternate address space word accesses with an ASI value of 0x4. *Table 5-12* gives the address map for these registers.

Table 5-12 Address Map for MMU Registers

VA[12:08]	Register
00	Processor Control Register
01	Context Table Pointer Register
02	Context Register
03	Synchronous Fault Status Register
04	Synchronous Fault Address Register
05-0F	Reserved
10	TLB Replacement Control Register
11–12	Reserved
13	Synchronous Fault Status Register[1]
14	Synchronous Fault Address Register**
15–1F	Reserved

1. Registers are cleared on read when this address is used

VA bits [31:13] are zero. VA bits [07:00] are ignored and should be set to zero by software. The use of a second access mode for the Synchronous Fault registers is provided as a diagnostic function (VA[12:08] = 0x13, 0x14). See the register description for details.

5.7.1 Processor Control Register

The Processor Control Register contains control and status bits for the microSPARC-IIep processor. The BM, IE, DE, and EN bits receive both the normal reset and watchdog resets (BM is set, IE, DE, and EN are reset). It is highly recommended that STAs to the PCR are immediately followed by 10 NOP instructions to keep the machine in a very consistent state. The PCR is defined in *Figure 5-9*.

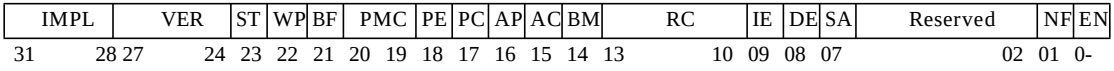


Figure 5-9 Processor Control Register

Field Definitions:

- Reserved (Rsvd / Rsv) — Bits [06:02] are unimplemented, should be written as zero and are read as zero.

- Implementation (IMPL) — The implementation number of this SPARC Reference MMU. This field is hardwired to 0x0 and is read only.
- Version (VER) — The version number of this SPARC Reference MMU. This field is hardwired to 0x4 (read only).
- Software Tablewalk enable (ST) — This bit enables the instruction_access_MMU_miss and data_access_MMU_miss traps for software instruction and data table walking respectively.
- Watch point enable (WP) — This bit enables the watch point trap. When set, it enables the Watch Point Trap logic within the MMU logic.
- Branch folding (BF) — This bit enables IU branch folding operation. When set enables the branch folding feature in the IU logic.
- Page Mode Control (PMC) — This bit enables the Page mode operation of the MMU/MEMIF interface. When it is set, the MMU's page mode registers track the usage of pages in memory to exploit page mode access to the DRAM when possible. Bit[19] controls page hit register 0, and Bit[20] controls page hit register 1. These bits are cleared on reset.
- Local (PCIC) Page Mode Control (AP) — This bit enables the Page mode operation of the local (PCI) bus interface. When it is set, the MMU's page mode registers track the usage of pages in Local Bus (PCI) space to exploit page mode access when possible. Page Mode operation is allowed to address space mapped to this PCI address space. Only PCI address space can be enabled with the Local Page Mode Control bit. This bit is cleared on reset.
- Parity Control (PC) — This bit controls the generation of parity (and checking on memory reads) in the memory interface as shown in *Table 5-13*.

  Parity is disabled with the PE bit.

Table 5-13 Parity Control Definition

PC	Meaning
0	Even Parity
1	Odd Parity

- Refresh Control (RC) — These four bits control the system DRAM refresh rate. For 100 MHz operation the RC field requires a 0x6 value. The RC field is defined in *Table 5-14*.

Table 5-14 Memory Refresher Control Definition

RFR_CNTL	Refresh Interval
0000	Every 128 MCLKs (down to 8.6 Mhz)
0001	No Refresh
0010	Every 704 MCLKs (down to 48 Mhz)
0011	Every 896 MCLKs (down to 60 Mhz)
0100	Every 1216 MCLKs (to 83 Mhz)
0101	Every 5120 MCLKs (low refresh)
0110	Every 1408 MClks (down to 100 Mhz)
0111	Every 1792 MClks (down to 125 Mhz)
1xxx	Reserved

- Boot Mode (BM) — This bit is set by both normal reset and watchdog reset and must be cleared for normal operation.
- Parity Enable (PE) — When set to one this bit enables word parity checking for all data entering the processor over the memory bus.
- Instruction Cache Enable (IE) — The instruction cache is enabled when this bit is set to a one. When it is zero, all references miss the cache. It is reset by both normal reset and watchdog reset.
- Data Cache Enable (DE) — The data cache is enabled when this bit is set to a one. When it is zero, all references miss the cache. It is reset by both normal reset and watchdog reset.
- Store Allocate (SA) — When set, this bit enables user store misses to be run in allocate mode. This means that if the page has been mapped as cacheable, the MMU signals the D-cache that a line fill must be done to satisfy the store miss. If the bit is cleared, the MMU disregards the page mapping information, and signals the D-cache that no line fill is required. The effect is that subsequent accesses to this data also “miss” in the D-cache if no allocate were done during the store miss. However, the amount of time the CPU is stalled is reduced when the allocate is not done. In either case the store data is placed in the store buffer, and subsequently to memory. No cache fill is done regardless of the line’s cacheability. The bit has no affect on Supervisor store misses. All Supervisor store misses are done in no-allocate mode. *Table 5-15* shows the possible settings.

Table 5-15 Store Allocate Setting

SA	User ST miss	Sys ST miss
0	No Allocate	No Allocate
1	Allocate	No allocate

- No Fault bit (NF) — When the NF bit is set, any access to an ASI, other than 8 or 9, that causes a fault is captured in the FSR and FAR, but no fault is generated to the processor. Faults resulting from access to ASI 8 or 9 are handled as normal regardless of the setting of this bit. Normal operation occurs while this bit is cleared.
- MMU Enable (EN) — When this bit is set to a one the MMU is enabled and translation occurs normally. When it is not set the physical address is forced to the 31 least-significant bits of the virtual address. This bit is reset by both normal reset and watchdog reset.
- Alternate Cacheability (AC) — When set, this bit specifies that the caches are enabled by the IE and DE bits even with the MMU disabled. When not set, the caches are disabled when the MMU is disabled. This should not be used during boot mode accesses. The access privilege associated with memory operations done under alternate cacheability mode is hardwired to ACC = 0x011 (User R/W/E and Sys R/W/E). Alternate Cacheability is a diagnostic feature that allows the caches to be enabled by the IE and DE bits even with the MMU disabled. When not set, the caches are disabled when the MMU is disabled. This should not be used during boot mode accesses. Instruction accesses work well with alternate cacheability when the accesses are to main memory space.

5.7.2 Context Table Pointer Register

The context table pointer register (CTPR) contains the base address of the context table. It is defined in *Figure 5-10*.

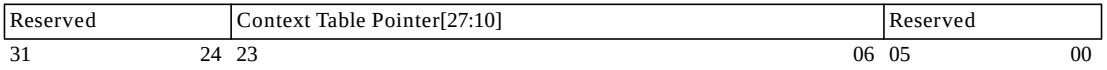


Figure 5-10 Context Table Pointer Register

The Context Table Pointer is 18 bits wide. The reserved fields are unimplemented, should be written as zero, and read as a zero.

5.7.3 Context Register

The context register (CXR) indexes into the context table. It is defined in *Figure 5-11*.

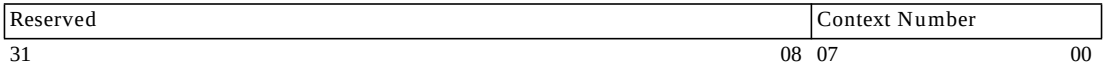


Figure 5-11 Context Register

The Context Register defines which virtual address space is considered the “current” address space. Subsequent accesses to memory through the MMU are translated for the current address space. This continues until the CXR is changed. The physical address of the root pointer is obtained by taking bits [23:06] from the CTPR to form mm_pa[27:10] and bits [07:00] from the CXR to form mm_pa[09:02]. mm_pa[30:28,01:00] are zero. Bits [31:08] of the CXR are unimplemented, should be written as zero, and read as zero.

5.7.4 Synchronous Fault Status Register

The Synchronous Fault Status Register (SFSR) provides information on exceptions (faults) issued by the MMU during CPU type transactions. There are three types of faults: instruction access faults, data access faults, and translation table access faults. If another instruction access fault occurs before the fault status of a previous instruction access fault has been read by the IU, the latest fault status is written into the SFSR and the OW bit is set. If multiple data access faults occur only the status of the one taken by the IU is latched into the SFSR (and address in the SFAR). If data fault status overwrites previous instruction fault status the OW bit is cleared since the fault status is represented correctly. An instruction access fault does not overwrite a data access fault. If a translation table access fault overwrites a previous instruction or data access fault the OW bit is cleared. An instruction access or data fault does not overwrite a translation table access fault. Reading the SFSR using ASI 0x4 and VA[12:08]= 0x03 clears it. However, reading the SFSR with VA[12:08]=0x13 does not clear it. Writes to the SFSR using ASI 0x4 and VA[12:08]=0x03 have no effect, while writes using VA[12:08]=0x13 update the register. The SFSR is only guaranteed to be valid after an exception is signalled. In other words, it may not be valid if there is no exception.

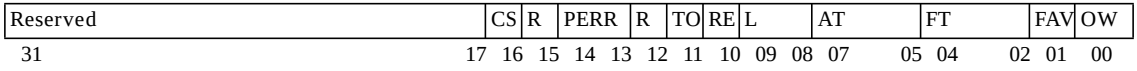


Figure 5-12 Synchronous Fault Status Register

Field Definitions:

- Reserved (Rsvd) (R) — Bits [31:17, 15, 12] are not implemented, should be written as zero, and read as zero.
- Control Space Error (CS) — This bit is asserted on the following conditions:
  - invalid ASI space
  - invalid ASI size
  - invalid VA field in valid ASI space and

- invalid ASI operation (for example a swap instruction to an ASI other than 0x8-0xB,0x20).

Note that the AT field is not valid on Control Space Errors.

- Time Out (TO) — A Time Out results from a CPU-initiated read transaction from an unsupported address space.
- Read Error (RE) — An error indication is returned on a CPU initiated read transaction from an unsupported address space.
- Parity Error (PERR) — The Parity Error[1:0] bits are set for external memory bus parity errors on the even and odd words respectively from memory.
- Level (L) — The Level field is set to the page table level of the entry which caused the fault. If an error occurs while fetching a page table (either a PTP or PTE) this field records the page table level for the entry. The level field is defined in *Table 5-16*.

Table 5-16 SFSR Level Field

L	Level
0	Entry in Context Table
1	Entry in Level 1 Page Table
2	Entry in Level 2 Page Table
3	Entry in Level 3 Page Table

- Access Type (AT) — The Access Type field defines the type of access which caused the fault. Loads and Stores to user/supervisor instruction space can be caused by load/store alternate instructions with ASI = 0x8–0xB. The AT field is defined in *Table 5-17*. Note that this field is not valid on Control Space Errors.

Table 5-17 SFSR Access Type Field

AT	Access Type
0	Load from User Data Space
1	Load from Supervisor Data Space
2	Load/Execute from User Instruction Space
3	Load/Execute from Supervisor Instruction Space
4	Store to User Data Space
5	Store to Supervisor Data Space
6	Store to User Instruction Space
7	Store to Supervisor Instruction Space

- Fault Type (FT) - The Fault Type field defines the type of the current fault. The FT field is defined in *Table 5-18*.

Table 5-18 SFSR Fault Type Field

FT	Fault Type
0	None
1	Invalid Address Error
2	Protection Error
3	Privilege Violation Error
4	Translation Error
5	Access Bus Error
6	Internal Error
7	Reserved

Invalid address errors, protection errors, and privilege violation errors depend on the AT field of the SFSR and the ACC field of the corresponding PTE. The errors are set according to *Table 5-19*.

Table 5-19 Setting of SFSR Fault Type Code

AT	FT Code								
	PTE[V]=0	PTE[V]=1(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	2	3	3
5	1	2	-	2	-	2	-	2	-
6	1	2	2	2	-	2	2	3	3
7	1	2	2	2	-	2	2	2	-

A translation error code (FT=4) is set when a SFSR PE type error occurs while the MMU is fetching an entry from a page table, a PTP is found in a level 3 page table, or a PTE has ET=3. The L field records the page table level at which the error occurred. The PE field records the word(s) having a parity error, if any. The protection error code (FT=2) is set if an access is attempted that is inconsistent with the protection attributes of the corresponding PTE. The privilege error code (FT=3) is set when a user program attempts to access a supervisor only page. An access bus error code (FT=5) is set when the SFSR PE field gets set on a memory operation that

was not a table walk. Additionally, this error code is also set on an alternate space access to an unimplemented or reserved ASI or the memory access is using a size prohibited by the particular type of ASI. If multiple errors occur on a single access the highest priority fault is recorded in the FT field (see below). If a single access causes multiple errors, the fault type is recognized in the priority given in *Table 5-20*.

Table 5-20 Priority of Fault Types on Single Access

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

- Fault Address Valid (FAV) — The Fault Address Valid bit is set if the contents of the Synchronous Fault Address Register (SFAR) are valid. The SFAR is valid for data exceptions and data errors.
- Overwrite (OW) — The Overwrite bit is set if the SFSR has been written more than once to indicate that previous status has been lost since the last time it was read. Overwrite status for combinations of error occurrences is given in *Table 5-21*.

Table 5-21 Overwrite Operations

Pending Error	New error	OW Status	Action Signalled
Translation Error	Translation Error	Set	Translation Error
Translation Error	Data Access Exception	Unchanged	Data Access Exception
Translation Error	Instruction Access Exception	Unchanged	Instruction Access Exception
Data Access Exception	Translation Error	Clear	Translation Error
Data Access Exception	Data Access Exception	Set	Data Access Exception
Data Access Exception	Instruction Access Exception	Unchanged	Instruction Access Exception
Instruction Access Exception	Translation Error	Clear	Translation Error
Instruction Access Exception	Data Access Exception	Clear	Data Access Exception
Instruction Access Exception	Instruction Access Exception	Set	Instruction Access Exception

5.7.5 Synchronous Fault Address Register

The Synchronous Fault Address register (SFAR) records the 32-bit virtual address of any data fault or translation reported in the SFSR. The SFAR is overwritten according to the same policy as the SFSR on data faults. Reading the SFAR using ASI 0x4 and VA[12:08] 0x04 clears it. Using VA[12:08] 0x14 to read the SFSR does not clear it. Writes to the SFAR using ASI 0x4 and VA[12:08] 0x04 have no effect while writes using VA[12:08] 0x14 update the register.

Note – Note that the SFAR should always be read before the SFSR to ensure that a valid address is returned. The structure of this register is shown in *Figure 5-13*.

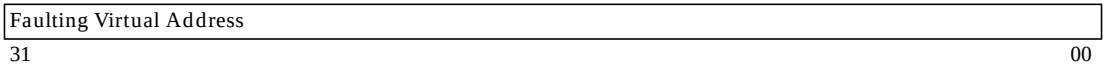


Figure 5-13 Synchronous Fault Address Register

5.7.6 TLB Replacement Control Register

The TLB Replacement Control register (TRCR) contains the TLB Replacement Counter and counter disable bit. The TRCR can be read and written using alternate load / store (LDA and STA) at ASI 0x4 with VA[12:08]=0x10. It is defined as follows.

Reserved			PCISP		BM SEL		VP	Reserved			PL			R	WP						TC	TLBRC					
31		25	24	23	22	21	20	19		17	16		14	13	12					07	06	05					00

Figure 5-14 TLB Replacement Control Register

Field Definitions:

- Reserved (R) — Bits [31:25, 19:17, 13] are unimplemented, should be written as zero and may be read as zero or one.

Note – In the MicroSPARC-II CPU, the Memory Speed (MEMSP) setting was able to be read in bit positions 22–21. In the microSPARC-IIep, the Memory Speed setting can be read from the MID register, described in this MMU section. Bit positions 22–21 are now used to read the boot mode.

- Boot Mode Select (BM_SEL) — These bits are used to indicate which boot mode has been selected by hardwired pin values (BM_SEL). The locations to boot from are encoded as shown in *Table 5-22*:

Table 5-22 Boot Mode Select (BM_SEL)

BM_SEL[1:0]	Boot From:
00	32 Bit Flash-Prom, on Memory Data Bus
01	8 Bit Flash-Prom, on Memory Data Bus
10	PCI Bus, Addresses Fooo.000 – F0FF.FFFF
11	PCI Bus, Addresses FFFE.000 – FFFF.FFFF

Note – When the PCI bus is selected for boot mode, the flash PROM interface is still available to the processor (physical address space 0x2), and defaults to the 32-bit Flash-PROM mode on the Memory Data Bus.

- PCI Bus Speed (PCISP) — These bits are used to indicate the divide by speed used to generate the PCI Bus clock (up to 33 MHz) from the internal CPU clock. This speed select value is defined by a hardwired pin values (div_ctl). The value is encoded as shown in *Table 5-23*.

Table 5-23 PCI Speed Select

PCI_SP	PCI Bus is CPU clock divided by:
0	Reserved
1	3 (up to 100 MHz CPU)
2	4 (up to 133 MHz CPU)
3	5 (up to 166 MHz CPU)

- Virtually tagged PTPs (VP) — This bit is used to enable the tagging of level-2 PTPs, in the TLB, with virtual tags instead of physical tags. When a table walk is started a check is made for a virtually tagged level-2 PTP before checking for a root level PTP. This check does not make the tablewalk any longer than usual. If a VPTP2 is found, the tablewalk goes directly to the level-3 PTE lookup. The TLB should be flushed after setting, or resetting, this bit to avoid mixing physically tagged level-2 PTPs and virtually tagged level-2 PTPs.
- TLB Replacement Counter Disable (TC) — The TLBRC does not increment when this bit is set.
- TLB Replacement Counter (TRC) — This is a 5-bit modulo-32 counter which is incremented by one during each CPU clock cycle to point to one of the TLB entries unless the TC bit is set. When a TLB miss occurs, the counter value is used to address the entry to be replaced.
- Wrap around Point (WP) — This 5-bit value is used to set a wrap around point for TLB replacement. For the microSPARC-IIep, this value should be set to wrap at 32 entries maximum. It may be set to a smaller value, to allow locked TLB entries.

- PTP Lock (PL) — This bit is used to enable PTP location limits. When this bit is set, PTP placement in the TLB is limited to entries 0–2. Bit[16] locks PTPs for level 2, Bit[15] locks PTPs for level 1, and Bit[14] locks PTPs for level 0. When PTP lock is used, the wrap point should not be set to a value less than 0x5. The MMU tries to use locations (entries) 3, 4, and 5 as alternate PTE stores when 0, 1, and 2 are reserved for PTPs. This use of locations 3, 4, and 5 is done without regard for the current setting of the WP.

5.8 MISC MMU Registers

The MISC MMU Registers contain controls for the remaining functions in the MMU.

IOPTE entries should not normally occupy any locations in the TLB. If they have been placed there by diagnostic operations, they may be flushed from the TLB by doing writes to the write-only Address Flush register (AFR). All of these internal MMU registers can be accessed directly by software using Control Space accesses with PA[30:24] = 0x10. Also, the entire TLB can be flushed using a control space access. The Control Space address map is shown in *Table 5-24*.

Table 5-24 MISC MMU, and Perf Counter Control Space

PA[30:00]	Device	R/W
1000 1000	Asynchronous (Memory) Fault Status Register	R/W
1000 1004	Asynchronous (Memory) Fault Address Register	R/W
1000 1050	Memory Fault Status Register	R/W
1000 1054	Memory Fault Address Register	R/W
1000 2000	MID Register	R/W
1000 3000	Trigger A Enables Register	R/W
1000 3004	Trigger B Enables Register	R/W
1000 3008	Assertion Control Register	R/W
1000 300C	MMU Breakpoint Control Register	R/W
1000 3010	Performance Counter A	R/W
1000 3014	Performance Counter B	R/W
1000 3018	VA Mask Register	R/W
1000 301C	VA Compare Register	R/W

Table 5-24 MISC MMU, and Perf Counter Control Space *(Continued)*

PA[30:00]	Device	R/W
1000 4000	Local Bus (IAFX) Queue Level	W
1000 6000	Local Bus (IAFX) Queue Level	R/O
1000 7000	Local Bus (IAFX) Queue Status	R/O

5.8.1 Asynchronous (Memory) Fault Status Register

The Asynchronous (Memory) Fault Status Register (AFSR) provides information on asynchronous faults during CPU initiated transactions to reserved address space and CPU write operations. This register is accessed using Control Space (0x10001000). A hardware lock is used to ensure that this register does not change while being read. Reading this register unlocks it. The bits of the AFSR are defined as follows:

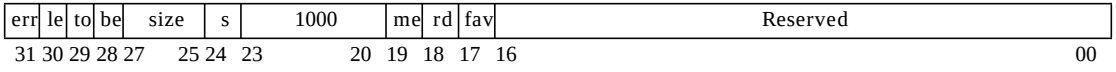


Figure 5-15 AFSR Register

Field definitions:

- [31]: Summary Error Bit (ERR) — One or more of LE, TO, or BE is asserted.
- [30]: Late Error (LE) — An error was reported after the transaction was done.
- [29]: Time Out (TO) — A write access timed out. An attempt to write reserved address space may result in this error.
- [28]: Bus Error (BE) — A write access received an error acknowledge. An attempt to write reserved address space may result in this error.
- [27:25]: Size (SIZE) — Reserved
- [24]: Supervisor (S) — CPU was in Supervisor mode when the error occurred.
- [23:20]: — Reserved; hard-wired to 0x1000
- [19]: Multiple Error (ME) — At least one other error was detected after the one shown.
- [18]: Read Operation (RD) — The error occurred during a read operation.
- [17]: Fault Address Valid (FAV) — The address contained in the AFAR is accurate and can be used in conjunction with the status in the AFSR. The only time the AFAR contents are invalid is on a late error.
- [16:00]: Reserved; not implemented; should be written as zero and read as zero

5.8.2 Asynchronous (Memory) Fault Address Register

The Asynchronous (Memory) Fault Address Register (AFAR) records the 31-bit physical address that caused the fault. This register is accessed using Control Space (0x10001004). Bit [31] should be written as zero and is read as zero. A hardware lock is used to ensure that this register does not change while being read. Reading the AFSR unlocks the AFAR. *Figure 5-16* shows the structure of this register.

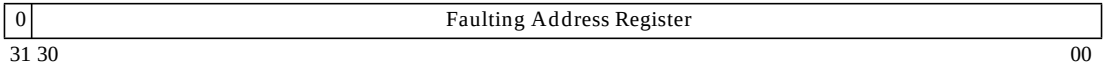


Figure 5-16 AFAR Register

Note that bit 31 is unimplemented, should be written as zero, and is read as zero. Also, this register is only held when an error is reflected in the AFSR.

5.8.3 Memory Fault Status Register

The Memory Fault Status Register (MFSR) provides information on parity faults. It is accessed using Control Space (0x10001050). This register is loaded on every request to memory unless it is locked. A hardware lock is used to ensure that this register does not change while being read if there were an error condition. Reading the register allows it to begin loading once again.

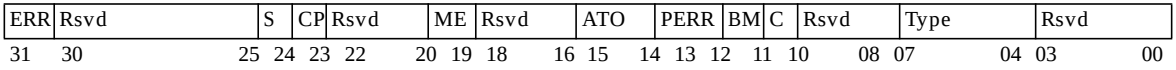


Figure 5-17 Memory Fault Status Register

Field Definitions:

- Reserved (Rsvd) — Bits [30:25, 22:20, 18:16, 10:08, 03:00] are not implemented, should be written as zero, and read as zero.
- Summary Error Bit (ERR) — One or more of PERR[1] or PERR[0] is asserted.
- Supervisor (S) — CPU was in Supervisor mode when the error occurred.
- CPU Transaction(CP) — The CPU initiated the transaction that resulted in the parity error.
- Multiple Error (ME) — At least one other error was detected after the one shown.
- Parity Error[1:0] (PERR) — These bits are set on external memory parity errors for the even and odd words (respectively) from memory. Parity errors can result from CPU or IO-initiated memory reads and byte or halfword (8 or 16 bit) write operations (which result in read-modify-writes).

- Local Bus (PCI) Timeout (ATO) — This bit is used to indicate that a time out has occurred for the current Local Bus operation.
- Boot Mode (BM) — This bit indicates that the error occurred while the PCR was indicating Boot Mode.
- Cacheable (C) — Address of error was mapped cacheable. In CPU initiated transactions this bit was from the C bit of the PTE, otherwise it is set to zero.
- Memory Request Type (Type[3:0]) — This field records the type of request that generated the parity error as shown in *Table 5-25*.

Table 5-25 Memory Request Type

Value(Hex)	Name	Definition
0	NOP	No memory operation
1	RD64	Read of 64 bits (2 words)
2	RD128	Read of 128 bits (4 words)
3	-	Reserved
4	RD256	Read of 256 bits (8 words)
5–8	-	Reserved
9	WR8	Write of 8 bits (1 byte)
A	WR16	Write of 16 bits (2 bytes)
B	WR32	Write of 32 bits (1 word)
C	WR64	Write of 64 bits (2 words)
D–F	-	Reserved

5.8.4 Memory Fault Address Register

The Memory Fault Address Register (MFAR) records the 31 bit physical address that caused the fault. This register is accessed using Control Space (0x10001054) and is loaded on every request to memory unless it is locked. A hardware lock is used to ensure that the register contents do not change during a read if there were an error condition. Reading this register allows it to begin loading once again. Bit [31] should be written as zero and is read as zero. *Figure 5-18* shows the structure of this register.

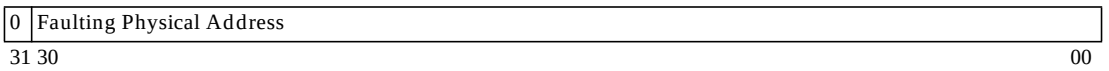


Figure 5-18 Memory Fault Address Register

Note that bit 31 is unimplemented, should be written as zero, and is read as zero. Also, this register is only held when an error is reflected in the AFSR.

5.8.5 MID Register

The MID register is used to control the miscellaneous functions of the microSPARC-IIep. This register can be accessed using IO MMU Control Space (0x10002000). The MID is defined in *Figure 5-19*.

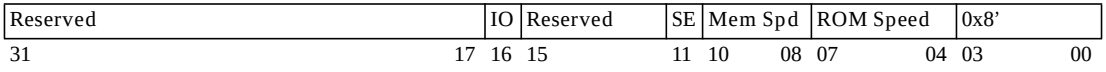


Figure 5-19 MID Register

Field definitions:

- Reserved — Bits [31:17, 15-12] are not implemented, should be written as zero, but may be read as zero or one.
- I/O Arbitration Enable (IO) — This bit is used to enable arbitration for the PCI interface to access the DRAM memory bus. This arbitration is between other internal usage of the DRAM memory bus, not between PCI devices. This bit must be set to a one to allow I/O access to the memory and is cleared to zero by reset.
- Standby Enable (SE) — This bit is used to enable the microSPARC-IIep to enter a power savings standby mode. While this bit is set, and there is no activity on the PCI bus, the processor stops execution and stops internal processor clocks. Refresh and PCI clocks are not affected by standby and continue. This Standby Enable bit is reset whenever there is PCI activity, including PCI DMA activity, which does not involve the processor. Therefore the Standby Enable bit should be set as part of a software idle loop. Refer to section on Chapter 10 for more information.
- Memory Speed (sp_sel[2:0], bits 10:08) — These bits are used to indicate the speed select being used for the DRAM memory interface. The values are encoded as shown in *Table 5-26*.

Table 5-26 Memory Speed Select

MEMSP sp_sel[2:0]	Fault Type
000	Up to 70 Mhz
001	Up to 85 Mhz
010	Up to 100 Mhz
011	Up to 133 Mhz
100–111	Reserved*

- ROM Speed. Bits[07–04] — These bits select the speed for read / write timing of the flash ROM. The Flash ROM interface supports devices that are compatible with the industry standard 28FxxxXX devices. This field is set to 0xF on reset. Refer to chapter 10 for more details on the Flash ROM operation. The ROM access time is set according to the relation:

((ROM Speed) - 1) X 3 X CPU cycle time = ROM access time*.

*If the ROM Speed is set to 0x0 or 0x1 the ROM access time used is 6 X CPU cycle time. These bits are R / W.

MID — This field is a constant 0x8 and is read only (writes to these bits are ignored).

Note – Endian control is performed using the Processor State register, and described in Compliance with SPARC Version 8.

5.8.6 Trigger A Enables Register

The Trigger A Enables register is used to define trigger events for Performance Counter A. Setting a field to “1” enables that trigger event for counting. This register can be accessed using Control Space (0x10003000).

Reserved	CO	R	R	R	FQ	FP	ST	MU	SU	SR	XL	MR	MC	MP	AB	MB	WB	DF	DS	DM	DH	IF	IS	IM	IH	OR	L
31 27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00

Figure 5-20 Trigger A Enables Register

Field definitions:

- Reserved (R) — Bits [31:27, 25:23] are not implemented and reserved, should be written as zero, and are read as zero.

- Trigger on edge or Level (L) — When set to a “1” this bit causes triggers defined in this register to be level sensitive; When cleared to a “0” the trigger becomes edge sensitive.
- OR — Combine triggers by ORing or ANDing function; When set to a “1”, this bit enables the Logical OR of the specified triggers to be counted. If the bit is set to a “0”, the triggers are logically ANDed to form the increment signal.
- I-cache miss (IH) — Asserted 1-cycle after the miss is detected, and sustained until the miss is resolved (ssparc.iwait_f)
- I-cache miss pending (IM) — Asserted 1-cycle after the miss is detected, and sustained until a corresponding memory request has been made (ssparc.ic_miss)
- I-cache streaming (IS) — Asserted after the first word has been fetched from memory, and until the cache line fill has completed (ssparc.ssparc_mmu.MMU_cntl.ic_stream)
- I-cache lookup (IF) — Instruction fetch from the Instruction cache
- D-cache miss (DH)— Asserted 1-cycle after the miss is detected, and sustained until the miss is resolved (ssparc.dwait_w)
- D-cache miss pending (DM) — Asserted 1-cycle after the miss detected and sustained until corresponding memory request is made (ssparc.dc_miss)
- D-cache streaming(DS) — Asserted after the 1st word has been fetched from memory, and until the cache line fill has completed (ssparc.ssparc_mmu.MMU_cntl.dc_stream)
- D-cache lookup (DS)— data read from the Data cache
- Write buffer full (WB)— Asserted while all four write buffer entries are valid (ssparc.dc_shold)
- Translation(XL) — 1-cycle pulse for each translation attempt (ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)
- Processor Tablewalk (SR) — Asserted for the duration of processor tablewalks; It can be used in conjunction with the translation count to determine TLB hit rate (ssparc.ssparc_mmu.MMU_cntl.sr_tw)
- Supervisor mode (SU) — Based on the processor PSR.S bit; can be used with other fields to determine supervisor overhead
- MMU breakpoint(MU) — Combined signal from MMU breakpoint decode
- Pipeline stalled (ST) — Asserted whenever the pipeline is stalled (1-cycle delay) (ssparc.iu_pipe_hold)
- Memory busy (MB) — Memory currently busy
- Local Bus (PCIC interface bus) busy (AB) — IAFX (Local Bus) interface currently busy
- Memory RMW op (MR)— Memory Read / Modify / Write operation requested
- Memory page mode access (MP) — Asserted once for each memory access that is on the same page as the previous access (for a given DRAM bank) (ssparc.mm_page)
- Memory precharge request (MC) — Asserted once for each memory access that indicated non-page hit prior to request (ssparc.mm_precharge)

- fhold_perf (FP) — FPU hold signal asserted for fld / fst dependency cases, or FP queue is full and another FP op is in the pipeline; guaranteed to hold the iu pipeline if psr.ef==1
- fhold_fq_full (FQ) — indicates that fhold is asserted because the FP queue is full and another FPop is in the pipeline
- Counter B_CO (CO)— Counter B Carry out; for trigger A only; this allows Counter A to reflect the number of counter B overflows

5.8.7 Trigger B Enables Register

The Trigger B Enables Register (see *Figure 5-21*) is used to define trigger events for Performance Counter B. Setting a field to "1" enables that trigger event for counting. This register can be accessed using Control Space (0x10003004).

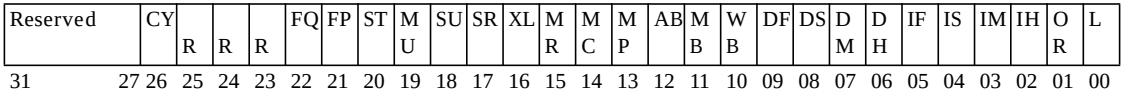


Figure 5-21 Trigger B Enables Register

Field definitions:

- Reserved (R) — Bits [31:27,25:23] are not implemented and are reserved. They should be written as zero, and are read as zero.
- Trigger on edge of Level(L) — When set to a "1" this bit causes triggers defined in this register to be level sensitive. When cleared to a "0" the trigger becomes edge sensitive.
- Combine triggers by ORing or ANDing function(OR) — When set to a "1", this bit enables the Logical OR of the specified triggers to be counted. If the bit is set to a "0", the triggers are logically ANDed to form the increment signal.
- I-cache miss (IH) — Asserted 1-cycle after the miss is detected and sustained until the miss is resolved (ssparc.iwait_f)
- I-cache miss pending (IM) — Asserted 1-cycle after a miss is detected and sustained until corresponding memory request is made (ssparc.ic_miss)
- I-cache streaming (IS) — Asserted after the 1st word has been fetched from memory and held until the cache line fill has completed. (ssparc.ssparc_mmu.MMU_cntl.ic_stream)
- I-cache lookup (IF) — Instruction fetch from the Instruction cache
- D-cache miss(DH) — asserted 1-cycle after the miss is detected, and sustained until the miss is resolved (ssparc.dwait_w)
- D-cache miss pending (DM) — asserted 1-cycle after miss is detected and sustained until a corresponding memory request is made (ssparc.dc_miss)

- D-cache streaming(DS) — asserted after the first word is fetched from memory and until the cache line fill has completed (ssparc.ssparc_mmu.MMU_cntl.dc_stream)
- D-cache lookup (DF) — Data read from the Data cache
- Write buffer full (WB) — Asserted while all four write buffer entries are valid (ssparc.dc_shold)
- Translation (XL) — 1-cycle pulse for each translation attempt (ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)
- Processor Tablewalk (SR) — asserted for the duration of processor tablewalks; can be used in conjunction with the translation count to determine TLB hit rate (ssparc.ssparc_mmu.MMU_cntl.sr_tw)
- Supervisor mode (SU) — based on the processor PSR.S bit. Can be used with other fields to determine supervisor overhead
- MMU breakpoint (MU) — combined signal from MMU breakpoint decode
- Pipeline stalled (ST) — asserted whenever the pipeline is stalled (1-cycle delay) (ssparc.iu_pipe_hold)
- Memory busy (MB) — memory currently busy
- Local Bus (PCIC Interface bus) busy (AB) — Local Bus (IAFX) interface currently busy
- Memory RMW op (MR) — memory Read / Modify / Write operation requested.
- Memory page mode access (MP) — asserted once for each memory access that is on the same page as the previous access (for a given DRAM bank) (ssparc.mm_page)
- Memory precharge request (MP) — asserted once for each memory access that indicated non-page hit prior to request (ssparc.mm_precharge)
- fhold_perf (FP) — FPU hold signal asserted for fld / fst dependency cases or FP queue is full and another FPop is in the pipeline; guaranteed to hold the IU pipeline if psr.ef==1
- fhold_fq_full (FP) — indicates that fhold is asserted because the FP queue is full and another FPop is in the pipeline
- Cycle count (FP) — always active

5.8.8 Assertion Control Register

The Assertion Control register (see *Figure 5-22*) can be used to invert any trigger event defined in the two trigger registers. Setting a field to “1” causes the trigger event for that field to be inverted prior to its entering the trigger register logic. This register can be accessed using Control Space (0x10003008).

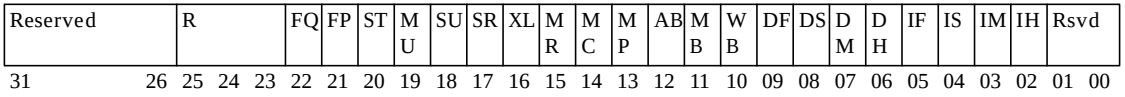


Figure 5-22 Assertion Control Register

Field definitions:

- Reserved (R) — Bits [31:23, 01:00] are not implemented, should be written as zero, and are read as zero
- I-cache miss (IH) — asserted 1-cycle after the miss is detected and sustained until the miss is resolved. (~ssparc.iwait_f)
- I-cache miss pending (IM) — asserted 1-cycle after miss detected and sustained until corresponding memory request is made (~ssparc.ic_miss)
- I-cache streaming (IS) — asserted after the first word is fetched from memory, and until the cache line fill has completed (~ssparc.ssparc_mmu.MMU_cntl.ic_stream)
- I-cache lookup (IF) — Instruction fetch from the Instruction cache
- D-cache miss (DH) — asserted 1-cycle after the miss is detected and sustained until the miss is resolved. (~ssparc.dwait_w)
- D-cache miss pending (DM) — asserted 1-cycle after miss detected and sustained until the corresponding memory request has been made. (~ssparc.dc_miss)
- D-cache streaming (DS) — asserted after the 1st word is fetched from memory and until the cache line fill has completed (~ssparc.ssparc_mmu.MMU_cntl.dc_stream)
- D-cache lookup (DF) — Data read from the Data cache
- Write buffer full inverted (WB) — (~ssparc.dc_shold)
- Translation inverted (XL) — (~ssparc.ssparc_mmu.MMU_cntl.r_tlb_used)
- Processor Tablewalk inverted (SR) — (~ssparc.ssparc_mmu.MMU_cntl.sr_tw)
- Supervisor mode inverted (SU)
- MMU breakpoint inverted (MU)
- Pipeline stalled inverted (ST) — (~ssparc.iu_pipe_hold)
- Memory busy inverted (MB)
- Local Bus (PCIC interface) busy inverted (AB)
- Memory RMW op inverted (AB)
- Memory page mode access inverted (AB) — (~ssparc.mm_page)
- Memory precharge request inverted (MC) — (~ssparc.mm_precharge)
- FPU pipeline hold inverted (FP)
- FPU Queue full inverted (FQ)

5.8.9 MMU Breakpoint Register

The MMU Breakpoint Register can specify a single breakpoint based on a number of field comparisons defined in the register. Any of the fields can be used as a stand alone compare or combined with other fields that are part of the MMU function. This register can be accessed using Control Space (0x1000300C).

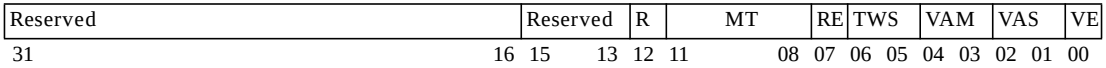


Figure 5-23 MMU Breakpoint Register

Field definitions:

- Reserved (R) — Bits [31:12] are not implemented and are reserved; should be written as zero and are read as zero
- Virtual Address Breakpoint enable (VE)
- Virtual Address Source (VAS) — these bits are used to control the source of the address to be used for breakpoint compare operations. The three bits are defined in *Table 5-27*.

Table 5-27 MMU Breakpoint Register VAS Field decode

VAS	Virtual Address Source
00	I-Cache Address
01	D-Cache Address (includes write buffer)
10	Reserved
11	Physical Address

- Virtual Address Memory operation (VAM) — these bits are used to describe the type of memory operation to be used together with the address compare for breakpoint. The possible memory operations for breakpoint definition are shown in *Table 5-28*.

Table 5-28 MMU Breakpoint Register VAM Field decode

VAM	Virtual Address Memory operation
00	disabled
01	read (D-Cache miss, or I-Cache miss)
10	write (D-Cache miss)
11	LDSTO

- Table Walk translation source (TWS) — these bits are used to describe the type of table walk operation to be used for the breakpoint. The bit definitions are shown in *Table 5-29*.

Table 5-29 MMU Breakpoint Register TWS Field decode

TWS	TableWalk translation Source
00	disabled
01	Instruction Fetch Table Walk
10	Data op Table Walk
11	Reserved

- Memory Request compare Enable (RE) — this bit field is used to enable breakpoint compare on the size and type information for memory operations. When used with the MT field this enable allows breakpoints on operations of a specific size to be enabled.
- Memory request Type (MT — this bit field is used to define the type and size of the memory operation for the breakpoint event. The possible definitions for this field are shown in *Table 5-30*.

Table 5-30 MMU Breakpoint Register MT Field decode

MT	Memory Request Type
0000	nop
0001	READ 8 bytes
0010	READ 16 bytes
0011	Reserved
0100	READ 32 bytes
0101–1000	Reserved
1001	WRITE 1 byte
1010	WRITE 2 bytes
1011	WRITE 4 bytes
1100	WRITE 8 bytes
1101	WRITE 16 bytes
1110–1111	Reserved

5.8.10 Performance Counter A

Performance Counter A is the first of two 32 bit counters. Counter A is incremented based on the assertion of triggers defined for counter A in the Trigger A Enables Register (see Section 5.8.6). Performance Counter A can be accessed using Control Space (0x10003010).

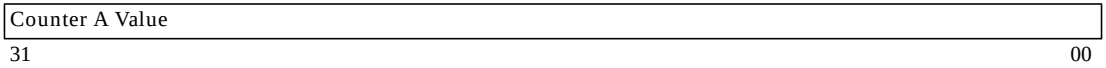


Figure 5-24 Performance Counter A

5.8.11 Performance Counter B

Performance Counter B is the second 32 bit counter. Counter B is incremented based on the assertion of triggers defined for counter B in the Trigger B Enables Register (see Section 5.8.7). Performance Counter B can be accessed using Control Space (0x10003014).

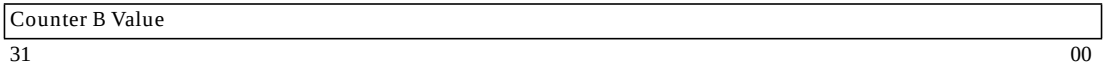


Figure 5-25 Performance Counter B

5.8.12 Virtual Address Mask Register

The Virtual Address Mask Register is used to disable the compare of specific bit fields in the Virtual Address Compare Register. Enables I1–I9 enable their respective fields for comparison. The N11 and N bits are used to decode the 'compare not' function. The N11 bit only affects the F field (VA[11]), and the N bit affects the range of VA[31:12]. When N=0, normal comparisons are made. When N=1, the compare result is inverted; so a 'hit' occurs when the addresses mismatch. This register can be accessed using Control Space (0x10003018).

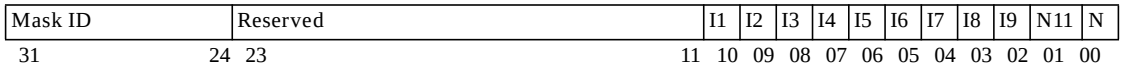


Figure 5-26 Virtual Address Mask Register

Field definitions:

- Reserved — Bits [31:11] are not implemented, should be written as zero, and are read as zero.
- N — Normal or inverted comparison enable; N=0 enables normal comparisons, N=1 enables inverted comparisons.
- N11 — Normal or inverted compare mode for bit 11 only
- I1 — Compare enable for VA[31:24]
- I2 — Compare enable for VA[23:18]
- I3 — Compare enable for VA[17:12]
- I4 — Compare enable for VA[11].
- I5 — Compare enable for VA[10:04]
- I6 — Compare enable for VA[03].
- I7 — Compare enable for VA[02].
- I8 — Compare enable for VA[01].
- I9 — Compare enable for VA[00].
- Mask ID — This read-only eight bit field is used to uniquely identify the revision level of the mask used to manufacture the part. The revision number is one of the entries found in *Table 5-31*.

Table 5-31 Mask ID

Mask ID	Mask Revision
0011 0100 (0x34)	1.0 First Tapeout
0011 0101 (0x35)	1.1 Revision
0011 0110 (0x36)	2.0 Tapeout

5.8.13 Virtual Address Compare Register

The Virtual Address Compare Register (*Figure 5-27*) is used to set the value of the virtual address that the breakpoint logic uses to compare against. This register should be used together with the Virtual Address Mask register to define the exact match criteria for the breakpoint. The VA can be either the I-cache or D-cache virtual address, or the address that is being translated by the MMU. Since the caches are virtually tagged, cache hit accesses do not need to be translated. Therefore, selects are provided to maintain address checking on a single source of address (regardless of hit/miss results). This register can be accessed using Control Space (0x1000301C).

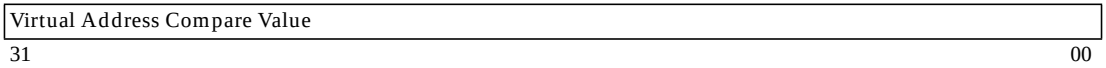


Figure 5-27 Virtual Address Compare Register

5.8.14 Local Bus (PCIC Interface) Queue Level Register

The Local Bus (PCIC Interface) Queue Level register is used to set the threshold value of the for CPU Local Bus (PCIC interface) read *holdoff*. If the number of operations in the queue is greater than the set threshold, CPU Local Bus (PCIC interface) operations are not issued. The CPU is held until such time as the Local Bus queue is at a level that would allow the operation to be issued. During this hold time the DVMA may freely access main memory. The threshold may only be set to 0x0, or 0x1. Note that this register is accessed through two different addresses. For writing this register, writes must be done to address 0x10004000. Writes to 0x10006000 do *not* update the register contents. For reading this register, reads must be done to address 0x10006000, reads to 0x10004000 do *not* return the current register contents. All reserved bits should be written as “0”, and are read as “0”. This register can be accessed using Control Space (0x10004000 for writes and 0x10006000 for reads).

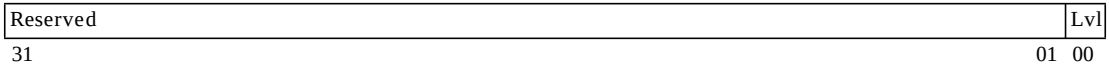


Figure 5-28 Local Bus Queue Level Register

5.8.15 Local Bus (PCIC Interface) Queue Status Register

The Local Bus (PCIC Interface) Queue Status register is a *read only* register that reflects the current number of Local Bus (PCIC interface) operations in the Local Bus queue. When read, this register should have a value of between 0x0 and 0x4. All reserved bits are read as “0”. This register can be accessed using Control Space (0x10007000).

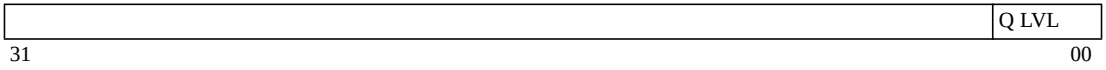


Figure 5-29 Local Bus Queue Status Register

5.9 Physical Address Register

The Physical Address Register (PAR) is used to hold translated physical addresses before they are used for either memory requests. This register cannot directly be read or written.See *Figure 5-30* for the structure of this register.

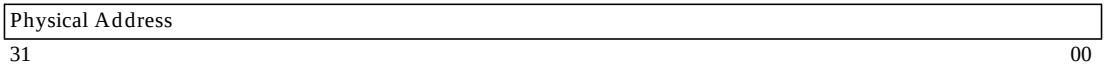


Figure 5-30 Physical Address Register

To avoid redundant retranslation, two extra registers are used to store PAR data. They are the Data Physical address register (DPAR) and the Instruction Physical Address register (IPAR).

5.10 TLB Table Walk

On a translation miss the table walk hardware translates the virtual address to a physical address by “walking” through a context table and from one to three levels of page tables. The first and second levels of these tables typically—but not necessarily—contain Page Table Pointers (PTP) to the next-level tables when accesses are due to CPU instruction or data addresses. A third level table entry should always be a page table entry (PTE) that points to a physical page to avoid a translation fault.

The table walk for a CPU-generated virtual address uses the context table pointer register (CTPR) as a base register and the context number contained in the context register (CXR) as an offset to point to an entry in the context table. The context table entry is then used as a PTP into the first level page table. At any address the table walk hardware finds either a PTE (which terminates its search) or a PTP. A PTP is used in conjunction with a field in the virtual address space to select an entry in the next level of tables. The table walk continues, searching through levels of tables as long as PTPs are found pointing to the next table. The table walk terminates when either a PTE is found or an exception is generated if a PTE is not found after accessing the 3rd level page table (or if an invalid or reserved entry is found). Note that PTPs and PTEs encountered during a table walk are not cached in the data cache. A full table walk is shown in *Figure 5-31*.

Note – The hardware table walking is done in big endian mode.

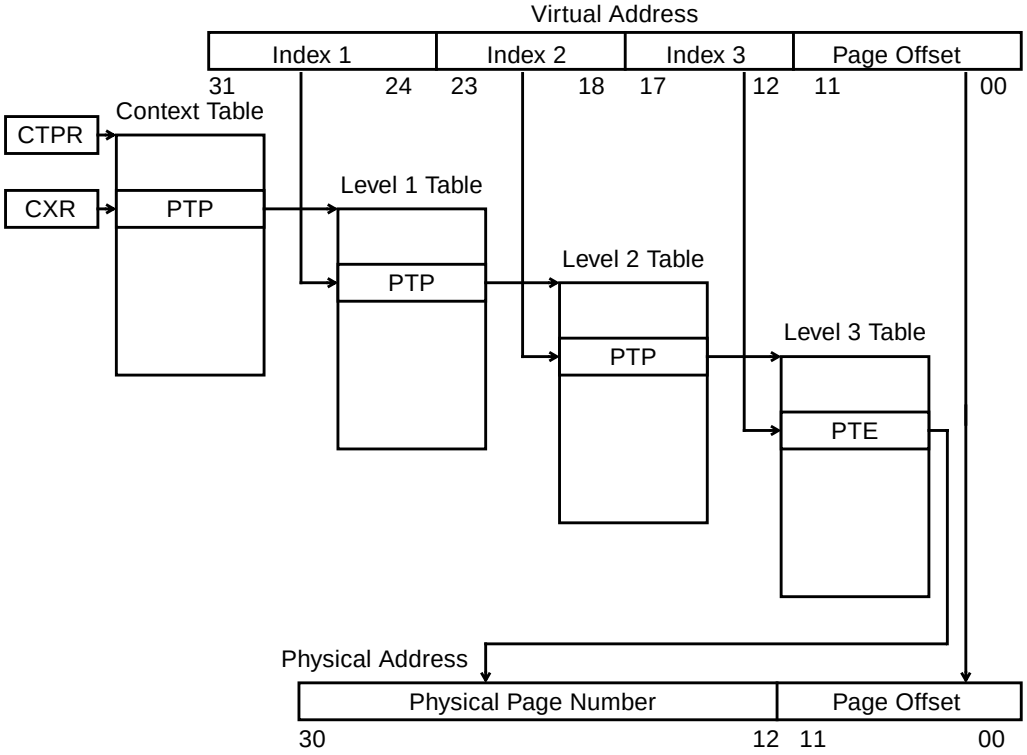


Figure 5-31 CPU Address Translation Using Table Walk

When the PTE is found, it is stored in a TLB entry, indexed by the TLBRC, and used to complete the original virtual to physical address translation. A table walk that is forced by a store operation to an unmodified region of memory causes the M bit in the PTE to be set. Any “entire” probe or normal tablewalk operation causes the R bit of the PTE to be set if it were not already.

The table walk mechanism is simplified when virtually-tagged level-2 PTPs are enabled. They are enabled by setting bit 20 of the TRCR (see Section 5.7.6, *TLB Replacement Control Register*). In this case, the MMU initially searches for the level-2 PTP by using the CXR, Index 1, and Index 2 of the virtual address. Should this PTP be found in the TLB, there is no need for the Context table and level-1 lookups, and the level-2 PTP can be used to look up the required PTE directly. This effectively halves the time required for the table walk. If the virtual-tagged PTP is not found in the TLB, the MMU starts the table walk from the context table.

The I/O address translations occur in the separate IOTLB in the PCI controller, which is managed by software. There are no IO PTEs placed in the CPU's TLB, and there are no operations performed on the CPU's TLB that are related to IO. IO translations are described in the PCIC section. See Section 9.5.6, *PCIC IOTLB Operation (DVMA)* and Section 9.5.7, *PCIC IOTLB Write Registers*.

5.11 Arbitration

The MMU block performs the primary memory arbitration function within the CPU. This is owing to the central nature of the MMU in the address flow of the machine. The different sources of memory activity are the:

- instruction cache block (for instruction fetches)
- data cache block (for loads and stores)
- TLB (during tablewalks and to keep the referenced and modified bits in the main memory page tables up to date)
- I/O DMA activity

The other entity needing main memory is the DRAM refresh logic. This function is folded into the arbitration scheme by the Memory Controller which must arbitrate between it and a request from the MMU.

The arbitrating requirements can be broken down into several different resource arbiters grouped into TLB arbitration and the internal memory bus arbitration.

5.11.1 TLB Arbitration

The current priority scheme places TLB references as highest priority, followed by data references, and finally instruction references. Tablewalks and updates to the memory PTEs due to changes to the Referenced and Modified bits are the highest priority. They imply that some other operation is in progress.

Table 5-32 TLB Reference Priority

Operation pending[1]		Results
IU Data Ref	Instr. Fetch	
Yes	X	Translate for IU Data Reference, Tablewalk if miss
No	Yes	Translate for Instruction Fetch, Tablewalk if miss

1. X = Don't Care,

5.12 Translation Modes

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

Table 5-33 Translation Modes

Name	ASI	Boot Mode	MMU En.	PA[30:00]
Boot IFetch	0x8, 0x9	Yes	X	PA[30:28]=0x2, PA[27:00]=VA[27:00] for flash boot PA[30:28]=0x3, PA[27:00]=VA[27:00] for PCI boot
Pass Through	0x8, 0x9	No	Off	PA[30:00]=VA[30:00]
Translate	0x8, 0x9	No	On	PA[30:12]=PTE[26:08], PA[11:00]=VA[11:00]
Pass Through	0xA, 0xB	X	Off	PA[30:00]=VA[30:00]
Translate	0xA, 0xB	X	On	PA[30:12]=PTE[26:08], PA[11:00]=VA[11:00]
Bypass	0x20	X	X	PA[30:00]=VA[30:00]

5.12.1 Page Hit Registers

The MMU is responsible for generating a signal to the memory controller indicating whether or not the current memory request can use page mode of the DRAMs. This is done by comparing the current physical address (mm_pa) against the physical addresses of the previous memory access to the even and odd memory banks. For this purpose the MMU has two registers which are used to store the current physical address: Register 0 is used if the memory operation is to an “even” bank or Register 1 is used if the memory operation is to an “odd” bank. The even or odd state of an address is determined by bit 25 of the physical address. For the page hit registers a *bank* refers to the physical address space consumed by a single DRAM SIMM module. Each register is used to store bits 30:12 of the physical address. If either register detects that the current access is to the same bank as the previous access the page mode signal to the memory controller is activated. This signal can be overridden by the control bits in the PCR register to disable this feature.

5.13 Errors and Exceptions

The MMU generates: instruction access error, instruction access exception, data access error, and data access exception for the SPARC IU. Also, an external interrupt is driven for asynchronous faults, which would indicate a level 15 interrupt. This interrupt must be enabled in the PCIC interrupt controller to be signalled to the CPU.

5.14 Diagnostic Features

All registers and RAM (and CAM) are accessible directly through alternate virtual address space loads and stores. There is also the capability to execute a breakpoint on certain conditions. This facility is set up through use of the various Breakpoint/ Performance counter registers.

5.14.1 Diagnostic Access of TLB

Diagnostic reads and writes to the 32 TLB entries are performed by using load and store alternate instructions in ASI 0x6 and the virtual address to select a particular TLB entry explicitly. The access must be a word access, all other data sizes result in an internal error. Depending on the virtual address specified, either the TLB Tag, or TLB PTE is referenced. The format for the TLB PTE is as described earlier. *Figure 5-32* gives the format of the Tag.

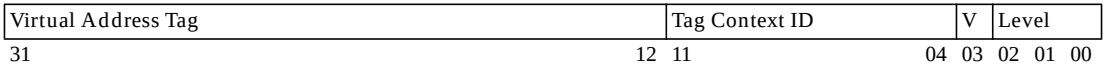


Figure 5-32 CPU Diagnostic TLB Upper Tag Access Format

Field Definitions:

- Virtual Address Tag — The 20 bit virtual address tag represents the most significant 20 bits (VA[31:12], the page address) of the virtual address being used. VA[11:00] is the byte within a page. The address in this field is physical when referencing physically tagged PTPs with the least-significant 20 bits containing PA[27:08].

- Context Tag — The 8-bit context tag comes from the value in the context register written by memory management software. Both it and the virtual address tag must match the CXR and VA[31:12] in order to have a TLB hit. This field contains a physical address {(PA[07:02]) + 2'b00}when referencing PTPs.
- Valid bit, Level bits — These three bits are used to enable the proper virtual tag match of root, region, and segment PTE's. The Valid bit indicates a valid entry. Bit 2 is the index bit for the root, bit 1 is the index for the region, and bit 0 is the index for the segment.

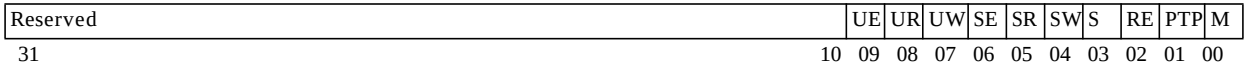


Figure 5-33 CPU Diagnostic TLB Lower Tag Access Format

Field Definitions:

- Reserved — These bits are not used and always return zero.
- UE — This bit indicates that the page has User Execute permission set.
- UR — This bit indicates that the page has User Read permission set.
- UW — This bit indicates that the page has User Write permission set.
- SE — This bit indicates that the page has Supervisor Execute permission set.
- SR — This bit indicates that the page has Supervisor Read permission set.
- SW — This bit indicates that the page has Supervisor Write permission set.
- Modified (M) — This bit is used to indicate that the page has been modified by a previous write operation.
- RE — Reserved; this bit is insignificant and is reserved
- Page Table Pointer (PTP) — This bit indicates that a PTP resides in this entry of the TLB. Note that all SRMMU flush types (except page) flush all PTPs from the TLB. This bit is insignificant for an ITLB Tag and is read as zero.

Note that when loading TLB entries under software control (using alternate space accesses) care should be taken to ensure that multiple TLB entries cannot map to the same virtual address. This may inadvertently occur when combining TLB entries that map different sizes of addressing regions. A level-3 PTE could be included in a TLB region for a level-1 or level-2 PTE, for example. The TLB output is not valid when this occurs.

Table 5-34 shows the virtual address is used to select the TLB entries:

Table 5-34 TLB Entry Address Mapping

Virtual Address	TLB Entry
0x0	Entry 0 PTE
0x4	Entry 1 PTE
:	:
0x78	Entry 30 PTE
0x7C	Entry 31 PTE
0x80–0xFC	Reserved
0x100	Entry 0 Tag[9:0]
0x104	Entry 1 Tag[9:0]
:	:
0x178	Entry 30 Tag[9:0]
0x17C	Entry 31 Tag[9:0]
0x180–0x2FF	Reserved
0x300	Entry 0 Tag[41:10]
0x304	Entry 1 Tag[41:10]
:	:
0x378	Entry 30 Tag[41:10]
0x37C	Entry 31 Tag[41:10]
0x380–FFFFFFFC	Reserved

5.14.2 MMU Breakpoint Debug Logic

The MMU breakpoint debug logic is intended for use in lab debug only since it requires setup through a scan facility. Its basic function is to stop the clocks when certain conditions occur. These conditions can be selected from a large number of conditions. The breakpoints which can be enabled are virtual address matching, virtual address source matching (includes type of request), memory request matching, tablewalk detection (includes type), and tablewalk level matching. A more detailed description and suggested pairings of these conditions follows.

A breakpoint can be entered on portions of the virtual address (the output of the virtual address multiplexing logic). These portions of the virtual address can be combined with other conditions to make their match conditions more case specific as shown in *Table 5-35*.

Table 5-35 Virtual Address Match Condition

Virtual Address Conditions	Conditions to be Paired With
VA[31:00]	ic_tlb & iu_fetch_f
VA[31:01]	(valid instruction fetch - D stage)
VA[31:02]	dc_tlb & read_w
[31:03]	(valid data read - W stage)
VA[31:12]	dc_tlb & write_w
VA[31:18]	(valid data write - W stage)
VA[31:24]	dc_tlb & ldsto_w
VA[10:02]	(valid data atomic op - W stage)
VA[11:02]	
!VA[31:12] & VA[11:02]	
!VA[31:11] & VA[10:02]	

A breakpoint can be invoked on the particular type of memory request being sent from the MMU to the MEMIF. This case is sampled when a memory request is actually being issued (mm_issue_req = 1). This condition can be paired with two other fields indicating the type of tablewalk occurring and the tablewalk level to match (if memory request indicates a tablewalk) as shown in *Table 5-36*.

Table 5-36 Memory Request Type

Name	Definition
NOP	No memory operation
RD64	Read of 64 bits (2 words)
RD128	Read of 128 bits (4 words)
RD256	Read of 256 bits (8 words)
WR8	Write of 8 bits (1 byte)
WR16	Write of 16 bits (2 bytes)
WR32	Write of 32 bits (1 word)
WR64	Write of 64 bits (2 words)
Tablewalk Type	
None	No tablewalk in progress
ic_tlb_tw	Tablewalk from instruction fetch
dc_tlb_tw	Tablewalk from data reference
Tablewalk Level	
Root Level	
Level 1	
Level 2	
Level 3	

5.14.3 Additional Features

There are other features that can be used for microSPARC-IIep debug. Some of these features are enabled using Processor Control register bits. Software tablewalks can be enabled by asserting PCR[23], the STW bit. For tablewalks in this mode—for instruction and data tablewalking respectively—the MMU causes the instruction_access_MMU_miss and data_access_MMU_miss traps to be done by software.

CHAPTER 6

Data Cache

6.1 Overview

The microSPARC-IIep data cache is an 8 kilobyte, direct-mapped cache that is used on load or store accesses from the CPU to cacheable pages of main memory. It is virtually-indexed and virtually-tagged. The write policy for stores is write-through with write allocate. The data cache is addressed by iu_dva[12:0] and it is organized as 512 lines of 16 bytes of data, each line of which has a cache tag associated with it. There is no sub-blocking. On a data cache miss to a cacheable location, 16 bytes of data are written into the cache from main memory.

Within the data cache block there are cache bypass paths. These paths are used for non-cached load references, and for streaming data into the integer unit and floating-point unit on cache misses.

A simple block diagram is shown in *Figure 6-1*.

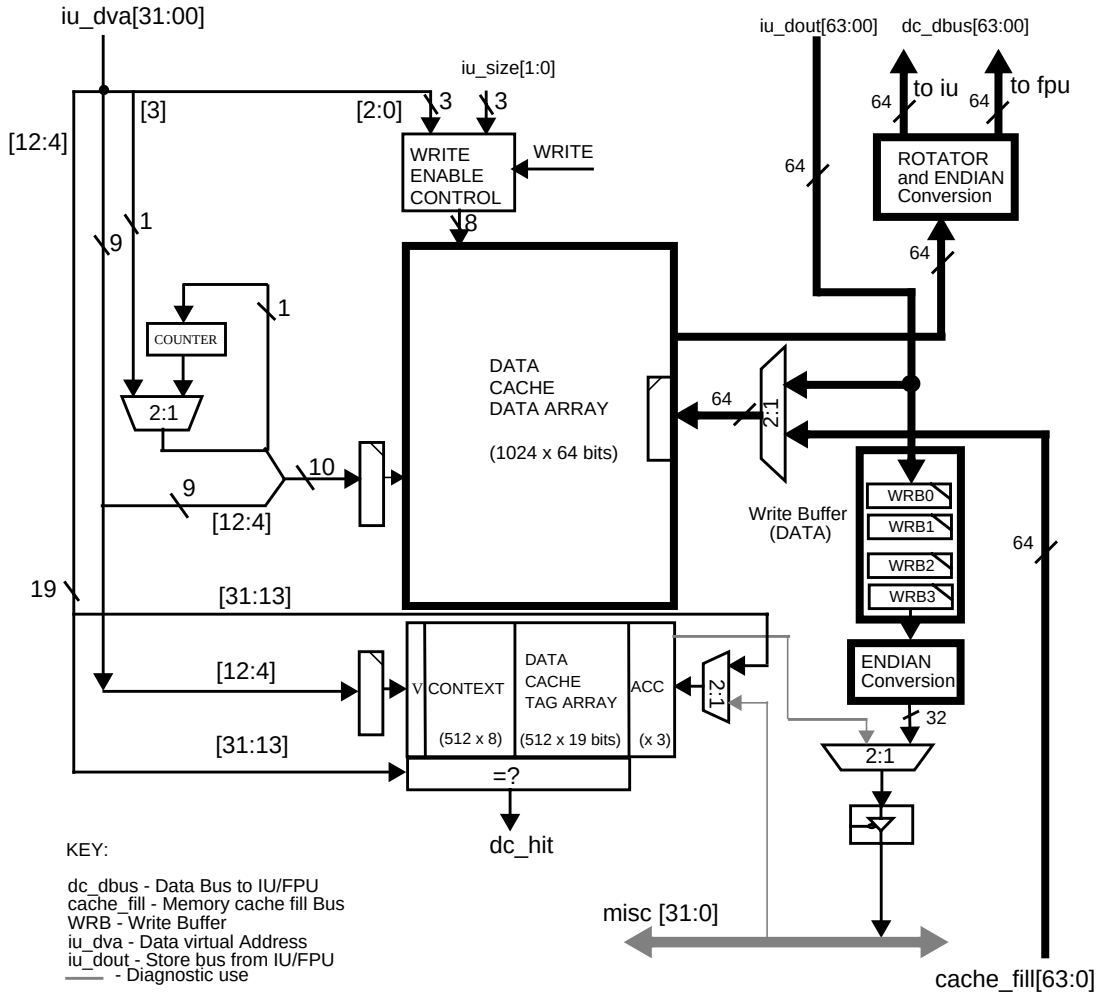


Figure 6-1 Data Cache Block Diagram

6.2 Data Cache Data Array

Since the data cache is a write-through cache, all write operations trigger an update of main memory. On cache misses, the missed cache line is read from memory into the cache (that is, write allocate). This avoids stale data remaining in the virtual data

cache owing to aliasing. This write-through with write allocate policy makes the data cache controller design more uniform than would be the case for a write-invalidate policy, since load misses are also handled in a similar way.

Diagnostic software can read and write the data cache directly by executing load or store alternate space instructions of *any size*, in ASI space 0xF. Virtual address bits VA[12:0] index the cache line in this mode (addresses roll-over to the proper cache line, modulo 512 cache lines). All other virtual address bits (addresses rollover), as well as the context bits, ACC bits and the valid bit are ignored during ASI=0xF operations.

There are two input sources to the data cache data array. The IU data_out bus (iu_dout) is used when the data cache is updated on an integer or floating-point store operation. The memory cache fill bus (cache_fill[63:0]) is used as input for fills on data cache misses, and also for diagnostic ASI [0xC, 0xD, 0xE] loads.

6.3 Data Cache Tags

A data cache tag entry consists of several fields as shown in *Figure 6-2*.

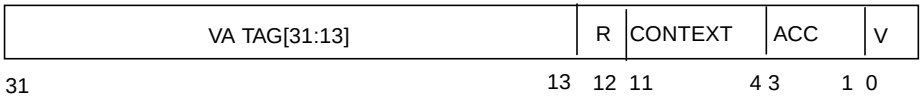


Figure 6-2 Data Cache Tag Entry

Field Definitions:

- [31:13]: Virtual Address Tag (VA TAG) — This field contains the virtual address of the data held in the cache line.
- [12]: Reserved (R) — Reserved.
- [11:4]: Context bits — These eight bits indicate the context of the particular cache line. They are filled from the TLB.
- [3:1]: Access (ACC) — These three bits indicate various levels of protection for that cache line. This field is copied from the TLB.
- [0]: Valid (V) — When set, the cache line contains valid data. This bit is set when a cache line is filled due to a successful cache miss; a cache line fill which results in a memory parity error leaves the valid bit unset. Stores to ASI space (0x10-0x14) conditionally clear the valid bits, as defined in *The SPARC Architecture Manual, Version 8*. See also Section 6.10, *Data Cache Flushing*.

- R bit — This bit is reserved (Not to be confused with the Reference bit in the TLB).

There are two input sources to the data cache tag array. The Virtual Address bits (DVA[31:13]) are used for cache tag updates on data cache misses. The miscellaneous bus (misc[31:0]) is used for store operations to ASI space (0xC, 0xD, 0xE) and to empty the store buffer contents to main memory.

Diagnostic software can read and write the data cache tags by executing only word-length LDA and STA (load and store alternate) instructions in ASI space 0xE. The virtual address bits [12:4] select one of the 512 tags; all other address bits are ignored.

Note – Due to different line sizes, the VA bits used to access the data cache are different from those used to access the instruction cache.

6.4 Write Buffers

The write buffers (WRB) are four, 64-bit registers in the data cache block used to hold data being stored from the IU or FPU to memory or other physical devices. WRB temporarily holds the store data until it is sent over the misc bus to the destination device. For halfword or byte stores, this data is left-shifted (with zero-fill) and replicated into proper byte alignment for writing to a word-addressed device (and the resulting word is replicated to make a 64bit doubleword), before being loaded into one of the WRB registers. There is no diagnostic read / write access to the WBR registers. The WRB is emptied prior to (actually before completion of) a load or store miss cache line fill sequence to avoid any stale data from being read in to the data cache on a miss to a virtual alias. Thus no snoop logic is needed to check for any data hazards between the WRB and the data coming back from main memory.

The address portion of the WRB contains virtual addresses rather than physical addresses. Thus the need for translation on store hits is avoided until the store is to be written to memory. The WRB is filled from the data cache controller and it is emptied by the MMU controller. There is an array of four valid bits associated with each entry of the WRB. On a store which traps, the WRB is properly flushed by the data cache controller, while the IU pipeline is held by that controller. This is necessary because the WRB is written at the end of the E-stage, and the store could trap in the W-stage of the pipeline.

The microSPARC-IIep CPU has a fifth bit for each write buffer. This bit holds the endian mode setting for the register at the time the entry is written.

6.5 Data Cache Fill

A 16-byte cache line is fetched from memory on a data cache miss. The requested doubleword is always returned first followed by the other doubleword, which wraps around a 16-byte boundary until the entire 16-byte block has been returned. The transfer rate is one doubleword every four to five cycles from memory (one double word, then 3 or 4 dead cycles)—see Section 5.8.5, *MID Register* for MID register and memory speed select for timing.

The cache array is written one clock cycle after each word appears on the cache_fill bus.*Table 6-1* illustrates the fill operation showing the order in which words are written into the cache.

Table 6-1 Data Cache Fill Ordering

Requested Word	Order of Fill
0	(0, 1), (2, 3)
1	(0, 1), (2, 3)
2	(2, 3), (0, 1)
3	(2, 3), (0, 1)

During the write cycles of a cache fill, data can be bypassed (or streamed) to the IU or FPU, one cycle after it appears on the cache_fill bus. During the dead cycles, data from any line in the cache can be written or read by subsequent load or store instructions.

On a cache miss for both loads and stores, the IU waits in the W-stage while the protections are being checked in the TLB. It resumes execution when the first requested word of the line is returned from memory.

6.6 ASI / STore Bus Interface

The data cache block interfaces to the misc bus for ASI ST / LD operations. Data from the data cache block to misc bus comes from the WRB. Control signals from the MMU and Memory Controller indicate when data is on the cache_fill bus, for loading into the data cache, and when data from the write buffer is to be driven onto the misc bus.

6.7 Cache Fill Bus Interface

The data cache is filled by a separate 64-bit bus, the cache_fill bus, from main memory. Because this bus is separated from other buses on the chip, it can be performance tuned without incurring a large chip area. In addition, this separation makes, it possible to use a 64-bit cache fill, rather than a 32-bit fill. The Cache Fill bus can also be bypassed directly (after being latched inside the Data RAM) to the IU and FPU, through a multiplexer in the data RAMs. This bypass path is used for streaming and for non-cached loads.

6.8 IU / FPU Data Bus Interface

The data cache block interfaces to an input and output IU/FPU data bus (iu_dout and dc_do). Data to the IU or FPU is either sourced from the latched output of the cache_fill bus (for streamed data on data cache misses, and for non-cached loads) or from the data cache (for data cache hits). Load data to the IU/FPU has to pass through a “rotator” block, which aligns the 64-bit word from memory or the cache to the IU/FPU. Data from the IU or FPU on store operations is loaded into the WRB and written into the cache RAMs. The interface to the FPU is 64 bits wide for both LD’s and ST’s, whereas the IU only has a 32 bit interface.

6.9 Endian Conversion

Two bits of the processor state register (PSR) control the endian conversion blocks. Refer to Section 3.13, *Compliance With SPARC Version 8* on page 40.

6.10 Data Cache Flushing

The data cache tags are implemented with all the five flush mechanisms (page, segment, region, context and user) as suggested in the SPARC Reference MMU appendix of *The SPARC Architecture Manual, Version 8*. These mechanisms are

activated by word size store instructions to ASI 0x10 - ASI 0x14. The *addressed* data cache line's (addressed by iu_dva[12:04]) valid bit is reset to zero by this operation, if it matches the flush address.

The store alternate flush using ASI 0x10 to ASI 0x14 flushes both the data cache and the instruction cache, although not necessarily in exactly the same clock cycle. Another way to flush both the caches is by explicitly writing a 0x0 into the valid bit of the cache line using the cache tag diagnostic ASIs 0xC for the instruction cache and 0xE for the data cache. Doing this resets the valid bit of the *addressed* cache line (addressed by iu_dva[12:04] in the data cache).

Note – The data cache is not flushed by the FLUSH instruction but the addressed instruction cache line is flushed.

A cache line is flushed if it meets the minimum criteria given in *Table 6-2*. S is the supervisor bit, U is the inverse of S, CNTXT is the matching of the context register and tag context, and VA[31:xx] is a comparison based on the virtual address tag.

Table 6-2 Flush Criteria for ASI 0x10-0x14

ASI[2:0]	Flush Type	Compare Criterion
0	Page	(S or CNTXT) and VA[31:12]
1	Segment	(S or CNTXT) and VA[31:18]
2	Region	(S or CNTXT) and VA[31:24]
3	Context	U and CNTXT
4	User	U
5, 6	reserved	-

6.11 Data Cache Protection Checks

The data cache tags also incorporate three access permission bits (ACC[2:0]) for checking access violations. These bits detect a protection or privilege exception in the W-stage, so that protection traps can occur in this stage. This decouples the virtually-addressed data cache from the TLB for a lot of cases. Load and store instructions that hit in the cache do not need the corresponding TLB entry to be present in the TLB although stores do need a translated physical address when they are ultimately drained from the WRB to main memory. If a store instruction creates a protection violation, the corresponding data cache line is invalidated. This action is necessary because the protection check signal is slower than the write to the data cache.

6.12 Cacheability of Memory Accesses

Pages that are declared as non-cacheable (C=0 in the PTE) are not cached in the data cache. For data consistency and implementation reasons, the following data are also not cached:

- Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU PCR=0). See Section 5.4, *Address Space Decodes* on page 69 for more information.
- Accesses while the data cache is disabled (DE bit of the MMU PCR=0). See Section 5.7.1, *Processor Control Register*.
- Accesses while alternate cacheability is disabled (AC bits of the MMU PCR=0).
- Accesses to any ASI except 0x8, 0x9, 0xA and 0xB.
- Accesses to any non-memory physical address (i.e., PA[30:28]) 0x1, 0x3, 0x4, 0x5, 0x6, 0x7). See Section 5.4, *Address Space Decodes* on page 69 for more information. Flash memory space (PA[30:28]=0x2) is cacheable.
- Accesses while in boot mode.
- Accesses by the MMU during tablewalks.

Note – An ST instruction to a non-cached address in ASI space 0x8, 0x9, 0xA and 0xB, invalidates the corresponding data cache line. This is because the ST has already updated the data RAM by the time the cacheable information is available. This information is usually in the MMU. for example, the TLB PTE.C bit.

Flash Prom space (PA[30:28]==0x2) may be cacheable.

6.13 Data Cache Streaming

When the first half of the data cache line is brought back from main memory, the IU pipeline is released by the data cache controller for both load and store instruction misses. During the period from the time the first half of the cache line is back until the second half of the cache line is filled, most instructions in the IU are allowed to proceed or stream, except in the following cases:

- LD/ST instructions to any ASI space other than 0x8 to 0xB.
- LD/ST instructions that access the second half of the *missed* cache line.
- Any instruction issued one cycle after a parity error is detected on a cache line fill.

- A store instruction issued one cycle before the second half of a line-fill cycle, due to resource conflict (both the IU and the data cache controller are trying to write the data-cache RAM).

The four-deep write buffer allows stores to continue execution during a cache miss. However, the pipeline is held if the write buffers become full during this streaming.

6.14 PTE Reference Bit Clearing

Many paging-based operating systems use the referenced bit (R bit) in the page table entry (PTE) to approximate least recently used (LRU) behavior in accessing frequently used pages quickly. Clearing the referenced bit of a PTE could be costly in the microSPARC-IIep CPU because clearing the R bit of a PTE entails flushing that page from the instruction and data caches, and the microSPARC-IIep CPU has virtual caches and no flush clear instruction. The cost of flushing is twofold:

- The cycles spent in flushing each line of the cache.
- The loss of cycles due to extra cache misses as a result of the cache line invalidations.

To avoid both of these problems, do not flush the instruction and data caches when the reference bit of PTE is reset.

This action could mean that some recently referenced pages get thrown out unnecessarily, but this should not happen often in practice. This scheme nearly approximates the original LRU behavior, and it works on the following paradigm:

If a page is frequently accessed, it generates *at least one* cache miss which is enough to make the MMU do a tablewalk (and hence set the R bit back to 1), before the operating-system daemon re-examines the R bit. This statement is based on the observation that, on average, if a page's R bit is reset, and this page is frequently accessed, there will be at least one cache miss to that page which forces the MMU to set the R bit back to 1 quickly.

6.15 Powerdown

The data cache RAM and tag RAM are both powered down to conserve energy during cycles when they are not used by the data cache controller. Powerdown is initiated by:

- The external standby pin

- The MID register bit
- The data cache controller state machine

Externally, the data cache controller follows a simple two-way handshake protocol of request/grant to go into powerdown mode. The data cache controller also holds the IU pipeline during this period. For more on this refer to Chapter 11, *Mode, Timing, and Test Controls.*

Internally the data cache controller goes into powerdown mode during various state machine states, when the data RAMs and tags are both not needed. This is because the RAMs and the tags both share the same powerdown control signal.

6.16 Diagnostic Strategy

Sublines and cache tags may be both read and written using ASI 0xF and 0xE respectively as previously discussed.

6.17 Parity Errors

Parity errors occurring on data cache line fill invalidate only that particular cache line being filled when the parity error occurred. Parity errors during non-cached misses do not cause any invalidations.

CHAPTER 7

Instruction Cache

7.1 Overview

The microSPARC-IIep instruction cache is a 16-kilobyte, direct-mapped cache. It is accessed on CPU instruction fetches from cacheable pages of main memory. It is virtually-indexed and virtually-tagged. The instruction cache is normally addressed by iu_iva[13:0] and is organized as 512 lines of 32 bytes of data. Each line has a cache tag store entry associated with it. There is no sub-blocking. On an instruction cache miss to a cacheable location, 32 bytes of data are written into the cache from main memory. The Instruction cache tags contain an 18-bit IVA[31:14] tag field, an 8-bit context field, 3-bit ACC field, and one valid bit.

Within the instruction cache block there are also cache bypass paths. These paths are used for non-cached instruction fetches, and for streaming instructions into the IU on a cache miss. A simple block diagram is presented in *Figure 7-1*.

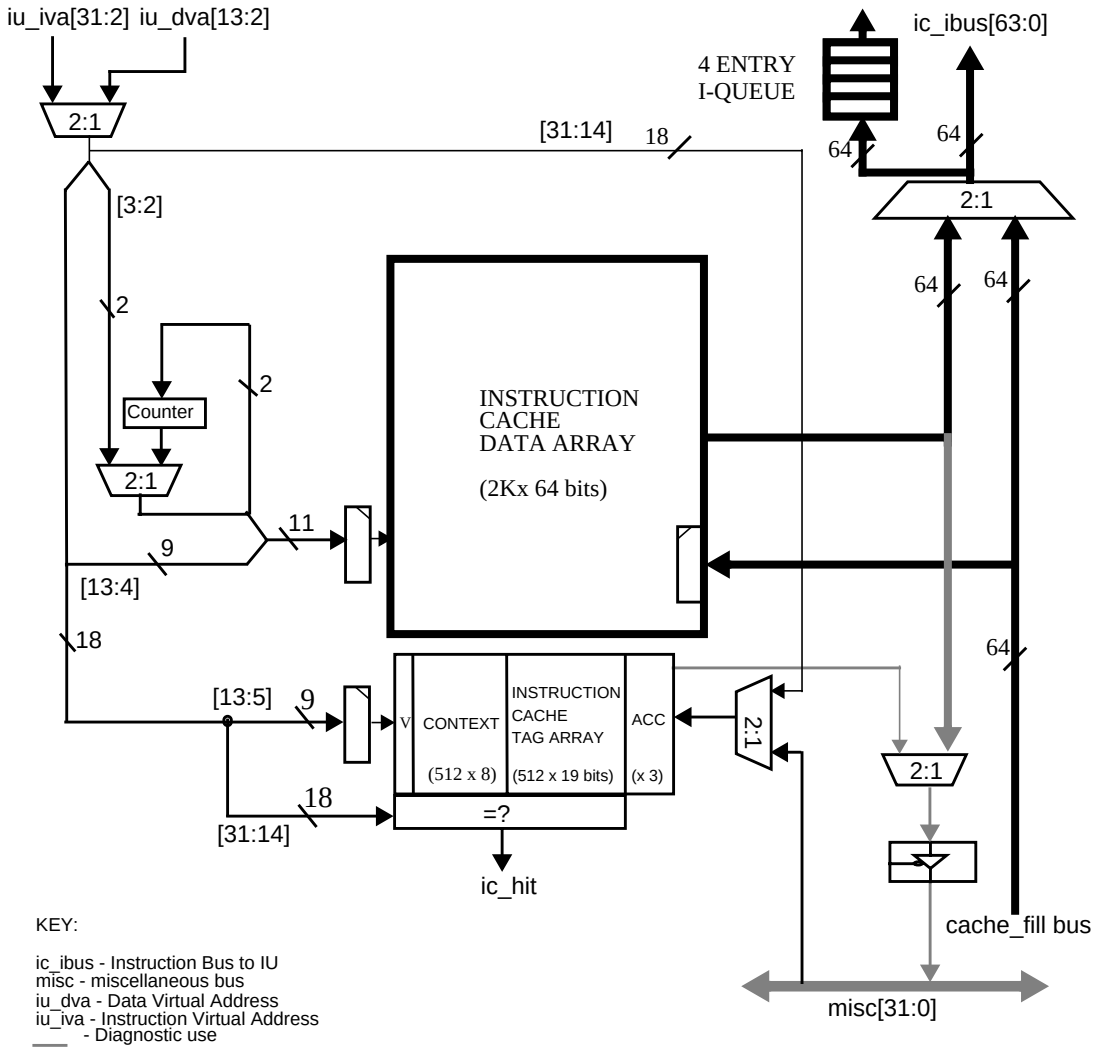


Figure 7-1 Instruction Cache Block Diagram

7.2 Instruction Cache Data Array

Diagnostic software may read and write the instruction cache directly by executing a single word load or store alternate space instructions in ASI space 0xD. Virtual address bits iu_dva[13:2] are used to address the instruction cache in this mode. All other virtual address bits (addresses rollover), as well as the Context bits, ACC bits and the Valid bit are ignored during these operations.

The internal misc[31:0] data bus is used as input/output for ASI operations, and the cache_fill[63:0] bus is used to fill the Instruction cache on instruction cache misses.

7.3 Instruction Cache Tags

An instruction cache tag entry consists of several fields shown in *Figure 7-2*.

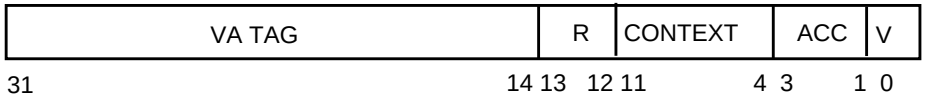


Figure 7-2 Instruction Cache Tag Entry

Field Definitions:

- [31:14]: Virtual Address Tag (VA TAG) — This field contains the virtual address of the data held in the cache line. The instruction cache controller writes this field from bits [31:14] of the virtual address (iu_iva) of the line.
- [13:12] — Reserved.
- [11:4]: Context bits — These indicate the context of the particular cache line. They are filled from the TLB.
- [3:1]: Access (ACC) bits — This 3-bit field indicates various levels of protection for that cache line. The field is copied from the TLB (see *Table 5-2* on page 66).
- [0]: Valid (V) — When set, the cache line contains valid instructions. This bit is set when a cache line is filled due to a successful cache miss; a cache line fill that results in a memory parity error leaves the valid bit cleared. A flush instruction clears the valid bit of the single line that is addressed by iu_dva[13:5] only if the tag for the addressed line matches the flush address. See Section 7.8, *Instruction Cache Flushing*.

There are two input sources for the instruction cache tag array. The virtual address bits needed for the tag are used for cache updates due to instruction cache misses. The misc bus is used as input for alternate store operations.

Diagnostic software can read and write the instruction cache tags by executing word-length LDA and STA (Load and Store Alternate) instructions in ASI space 0xC.; dva bits [13:5] select one of the 512 tags; all other address bits are ignored.

Note – Due to different line sizes, the VA bits used to access the instruction cache are different from those used to access the data cache.

7.4 Instruction Hit / Miss

Data is fetched from memory on instruction cache misses in 32-byte blocks. Memory returns 32 bytes of data, starting with the requested double word followed by the three remaining double words (even double word, then odd double word), which wraps around a 32-byte boundary until the entire 32-byte block is returned. The transfer rate is one double word every four or five cycles from memory (one doubleword, then 3/4 dead cycles). The cache array is written during the cycle that each word appears on the cache_fill bus[63:0]. *Table 7-1* illustrates the fill operation showing the order that words are written into the cache. Depending on the “sp_sel” memory speed selection setting of the microSPARC-IIep CPU, there is a gap of some (usually 3 or 4) internal clocks in between every two words filled into the cache.

Table 7-1 Instruction Cache Fill Ordering

Requested Word	Order of Fi.ll
0	(0, 1), (2, 3), (4, 5), (6, 7)
1	(1, 0), (2, 3), (4, 5), (6, 7)
2	(2, 3), (4, 5), (6, 7), (0, 1)
3	(3, 2), (4, 5), (6, 7), (0, 1)
4	(4, 5), (6, 7), (0, 1), (2, 3)
5	(5, 4), (6, 7), (0, 1), (2, 3)
6	(6, 7), (0, 1), (2, 3), (4, 5)
7	(7, 6), (0, 1), (2, 3), (4, 5)

During an instruction cache fill, instructions from the missing line can be supplied to the IU or FPU by means of two separate mechanisms, collectively called *streaming*. In the first type of streaming—bypass streaming—instructions are bypassed around the cache data array to the IU/FPU in the same cycle that the array is being written.This action can occur in all clock cycles of the fill sequence except the gap cycles. The second form, gap streaming, occurs only during the gap cycles. Any instruction word, from any line in the cache, that has already been written into the RAM array can be accessed by reading the array. In a given cycle, the IU is able to accept immediately the instruction word that it needs and instruction words that it may need in the future (prefetching). If, in a given cycle, the IU requests a word that is available via streaming, that word is supplied to the IU and the pipeline is allowed to advance. The concept of streaming does not apply to non-cached instructions, as the IU does not have to be held for a cache fill.

7.5 IASI Bus Interface

The instruction cache block interfaces to the misc bus for ASI operations. Data from the misc bus to the instruction cache comes from the write buffer. There are control signals from the MMU to indicate when data on misc[31:0] is to be loaded into the instruction cache.

7.6 ICache fill Bus Interface

The instruction cache is filled by a separate 64-bit bus, cache_fill[63:0], from main memory. Because this bus is separate from the other misc[31:0] bus on the chip, this bus can be tuned for higher performance. Using a 64-bit bus, rather than a 32 -bit one, gives the microSPARC-IIep CPU a shorter cache fill latency.

7.7 IU Instruction Bus Interface

The instruction cache block drives the ic_ibus, the IU instruction bus. Instructions to the IU or FPU are sourced from either the latched value of the cache_fill bus—for bypass-streamed instructions on instruction cache misses, and for non-cached instruction fetches—or from the instruction cache data array for instruction cache hits, and for dead-cycle streamed instructions on instruction cache misses. The IU also fills the I-queue from this bus(ic_ibus).

7.8 Instruction Cache Flushing

The instruction cache tags are implemented with all five flush mechanisms: page, segment, region, context and user, as suggested in the *SPARC Reference MMU Architecture* appendix of *The SPARC Architecture Manual, Version 8*. They are activated by word length alternate store instructions to ASI=0x10 to ASI=0x14. The IFLUSH instruction also can be used to flush the instruction cache. In both cases, the *addressed* (by iu_iva[13:05]) instruction cache line's valid bit is reset if the corresponding tags match. The match criterion is determined by the type of flush instruction. The instruction queue is not flushed on an instruction cache flush because the maximum depth of the instruction queue is only four instructions and the IU disables any more instruction fetches when it decodes an instruction cache flush opcode in the D-stage. *The SPARC Architecture Manual, Version 8* allows five instructions after an instruction cache flush instruction, for the IU to make the pipeline, instruction queue, and instruction cache consistent. For the data cache, the instruction tag diagnostic ASI=0xC can be used to reset the valid bit.

It is recommended that the instruction cache be flushed whenever the referenced bit (R bit) of any cacheable line is reset in the corresponding entry in the page tables.

Note – To maintain consistency, software must flush the instruction cache whenever the ACC bits or the C bit of a cacheable location is changed in the corresponding entry in the page tables.

A cache line is flushed if it meets the minimum criteria given in *Table 7-2*, where S is the supervisor bit, U is the inverse of S, CNTXT is the matching of the context register and tag context, and VA[31:xx] is a comparison based on the virtual address tag.

Table 7-2 Flush Criteria for ASI 0x10–0x14

ASI[2:0]	Flush Type	Compare Criterion
0	Page	(S or CNTXT) and VA[31:12]
1	Segment	(S or CNTXT) and VA[31:18]
2	Region	(S or CNTXT) and VA[31:24]
3	Context	U and CNTXT
4	User	U
5, 6	reserved	–

7.9 Cacheability of Memory Accesses

Pages that are declared as non-cacheable (C=0 in the PTE) are not cached in the instruction cache. For data consistency and implementation reasons, the following instruction fetch operations are not cached regardless of the state of the PTE.C bit.

- Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU PCR=0). Refer to Section 5.7.1, *Processor Control Register*.
- Accesses while the instruction cache is disabled (IE bit of the MMU PCR=0). Refer to Section 5.7.1, *Processor Control Register*.
- Accesses that occur while in boot mode.
- Accesses to sources in physical address spaces 1-7. See Section 5.4, *Address Space Decodes* for more information. Flash memory space (PA[30:28]=0x2) is cacheable.

Note – Flash PROM space (PA[30:28]==0x2) may be cacheable.

7.10 Diagnostic Strategy

Sublines and cache tags may both be read and written using ASI 0xD and 0xC respectively.

CHAPTER 8

Memory Interface

8.1 Overview

The memory interface provides tight coupling between the processor core and the external memory. The important features include:

- Support for 256 megabytes of system DRAM
- 64-bit data bus to increase memory bandwidth.
- 1-bit parity per word (32 bits) for reduced cost; parity checking can be disabled.
- Support for different density devices by dividing memory into blocks. This allows relatively small memory increments with a small number of blocks.
- Usage of compatible EDO DRAM that meets fast-page mode DRAM timing is allowed (provided it meets the conditions described in Section 8.2.
- Support for dual-RAS and single-RAS modes. In dual RAS mode even and odd RAS can be active together.
- Support for the next generation of DRAM devices by allowing for future higher memory requirements.

The microSPARC-IIep Memory Interface block is logically divided into three subsections, the Memory Control Block (MCB), The Data Aligner and Parity check/Generation Logic (DPC) and the RAM Refresh Control (RFR).

Typically a carefully laid out system board using the microSPARC-IIep chip requires 60 ns, 3.3 V/5 V DRAMs at 100 MHz clock speed. however, the designer should use the memory interface AC specifications in the microSPARC-IIep datasheet to select the appropriate DRAM speed for a specific system and clock speed.

8.2 Memory Organization

The microSPARC-IIep architecture defines a 28-bit physical address space for memory (with PA[30:28] = 0x0). This supports a 256 megabyte block for system DRAM. See Appendix B, *Physical Memory Address Map*.

This 256 MB is divided into eight banks, each capable of addressing up to 32 megabytes. The banks are defined as follows:

- Each bank is selected by a separate RAS line. There is a total of eight RASs (RAS_L[7:0]) for eight DRAM banks.
- The banks have 64-bit data paths to the microSPARC-IIep CPU.
- Banks 0, 2, 4, 6 use the same 2-bit CAS lines (CAS_L[1:0]) to select the upper or lower 32 bits (high or low word).
- Banks 1, 3, 5, 7 use the other 2-bit CAS lines (CAS_L[3:2]) to select the upper or lower 32 bits (high or low word).
- All the banks use the same write signal (MWE_L) and same output enable (MOE_L, required for EDO RAMS only). Fast-page mode and EDO DRAMs cannot be mixed within the same system unless their output RAM enables can be connected to the microSPARC-IIep memory output enable pin MOE_L.
- All the banks use the same 22-bit multiplexed row/column address bus MEMADDR[11:0].

The memory interface is designed with the 4-bit wide DRAM devices in mind. To provide a 64-bit wide data bus, 16 such devices (or two SIMMs with eight devices on each) are required. Each bank requires two additional 1-bit wide devices of the same depth (if using SIMMs, one on each SIMM) to store the two parity bits. Hence, each bank can be populated using one of the configurations listed in *Table 8-1*.

Table 8-1 Memory Bank Population

Size of Data	Width of Data Bus	Configuration
8MB	64	16 1Mx4 devices for data and 2 1Mx1 for parity 2 1Mx33 SIMMs
16MB	64	8 2Mx8 devices for data and 2 2Mx1 for parity 2 2Mx33 SIMMs
32MB	64	16 4Mx4 devices for data and 2 4Mx1 for parity 2 4Mx33 SIMMs

8.2.1 Access to Unused or Unpopulated Memory Regions

If a bank contains less than the defined maximum of 32 megabytes, the real memory is mirrored on to the higher unused sections of the bank. Any access to the unused sections is mirrored to the corresponding location in the lowest block and no errors are generated. For example, if a bank contains 8-megabytes of real memory, this memory is mirrored on the remaining three empty 8 megabyte sections.

However, access to a completely empty bank results in undetermined data that may cause a parity error.

8.2.2 Dual-RAS Mode

Two basic modes of operation are supported as controlled by the SIMM32_SEL input pin:

- When this input pin is hardwired low, the memory interface operates in dual RAS mode. In this mode, an even and odd RAS are allowed to be active simultaneously. The CAS lines are also qualified by even or odd block. CAS_L[1:0] are qualified with even banks (physical address bit 25 = 0) and CAS_L[3:2] are qualified with odd banks (physical address bit 25 = 1). Using this technique, an even and odd RAS_L line can be active without conflict on the memory data bus. This mode is only supported with fast-page mode DRAMs in configurations of 16 MB SIMMs (or less) and 32 MB DIMMs (or less). See the dual-RAS mode configuration example in *Figure 8-4*. The two page-hit registers support page mode operations while under dual-RAS mode. See Section 5.12, *Translation Modes* on page 101 for more information about these two page-hit registers.
- When the SIMM32_SEL input pin is hardwired high, only a single RAS_L line is allowed to be active and CAS_L lines are not qualified by even or add block (CAS_L[1:0] and CAS_L[3:2] are logically identical). This allows support of EDO DRAMs—without performance improvement—and 32 MB SIMMs. This mode could result in up to a 5% performance loss. See the single-RAS mode examples in *Figure 8-5* and *Figure 8-6* for usage of 32-megabyte FPM SIMMs and 32 megabyte EDO DIMMs respectively.

Note – Any EDO memory module that has its output enable grounded internally is currently not supported as this results in a drive conflict on the memory data bus.

8.2.3 Address Mapping For System DRAM

From the 31 bits of the physical address bus driven by the MMU block (mm_pa[30:0]), the three MSBs (mm_pa[30:28]) represent one of the eight physical address spaces (PAS) as defined in microSPARC-IIep architecture. From these, only PAS0 is of concern to the MCB, since an MMU request from the MCB is only made if an access to system memory is required. Hence ADEL ignores the mm_pa[30:28] bits.

When a memory cycle request is detected (that is, PA[30:28] = 0x0), the address bits PA[27:02] are used to generate DRAM column and row addresses and control signals. *Table 8-2* describes the decode scheme used for system memory.

Table 8-2 Physical Address Decode for System Memory

PA	Decode
30-28	Not used. System memory limit is 256 MB.
27-25	Select 1 of 8 RASes (each bank is 32 MB): 000RAS_L[0]Bank 0 100 RAS_L[4] Bank 4 001RAS_L[1]Bank 1101 RAS_L[5] Bank 5 010RAS_L[2]Bank 2110 RAS_L[6] Bank 6 011RAS_L[3]Bank 3111 RAS_L[7] Bank 7
24	Row address bit 10 (MEMADDR[10]). Required for 16MBit DRAMs.
23	Column address bit 10 (MEMADDR[10]) and row address bit 11 (MEMADDR[11]). Required for 16MBit DRAMs. See text for more information.
22	Row address bit 9 (MEMADDR[9]). Required for 4MBit DRAMs.
21	Column address bit 9 (MEMADDR[9]). Required for 4MBit DRAMs and up.
20-12	Row address bits 8 to 0 (MEMADDR[8:0]). Required for 1MBit DRAMs and up.
11-3	Column address bits 8 to 0 (MEMADDR[8:0]). Required for 1MBit DRAMs and up.
2	Select one of 4 CASes: (only qualified with PA[25] when SIMM32_SEL = 0) 0CAS_L[0]Lower address word (MEMDATA[63:32]) for banks 0,2,4,6. (PA[25] = 0) 1CAS_L[1]Higher address word (MEMDATA[31:0]) for banks 0,2,4,6. (PA[25] = 0) 0CAS_L[2]Lower address word (MEMDATA[63:32]) for banks 1,3,5,7. (PA[25] = 1) 1CAS_L[3]Higher address word (MEMDATA[31:0]) for banks 1,3,5,7. (PA[25] = 1)
1-0	Not used for external decode. Byte and halfword writes are achieved by a MCB and DPC read, modify, write sequence. This bits are then used to select the appropriate data fields.

A maximum of 1024 memory cycles can be made from a contiguous block, while remaining within a DRAM page. This gives a maximum of 8K (1024x64) block size that can theoretically be accessed using page mode cycles only.

Table 8-2 shows the staggered decoding of PA[24:21] for MEMADDR[10:9]. This was necessary in order to allow different size devices (1Mx4 and 4Mx4) to be used while maintaining the largest common contiguous block, which is dictated by the least dense device.

Also, as shown in *Table 8-2*, PA[23] is used as both MEMADDR[10] for column address and MEMADDR[11] for row address. This supports two different 4Mx4 DRAM architectures, 11x11 matrix and 12x10 matrix.

The 4Mx33 SIMMs use the DRAMs based on 11x11 matrix (to allow the use of a 4Mx1 DRAM for parity). The microSPARC-IIep CPU also provides a 12th DRAM address bit, which allows the 12x10 matrix DRAMs to be used.

Note – Byte and half word writes are converted to read-modify-write sequences where the full word is read, updated with the byte or half word, and written beck to DRAM.

8.3 Memory Control Block (MCB)

The Memory Control Block (MCB) keeps track of the priorities of memory operations and completely controls the DRAM based main memory during the memory operations:

- data reads
- writes
- read-modified-writes required for CPU execution
- instruction fetches and prefetches
- translation buffer accesses during table walks
- reads and writes by IO devices
- all RAM refreshes

The MCB contains two major logic blocks, “ASM” and “ADEL” which perform memory arbitration and address mapping functions respectively. These blocks are described in the following subsections. The MCB also includes some input and output register blocks, which provide input and output signal synchronization.

A schematic diagram of the MCB is shown in *Figure 8-1*.

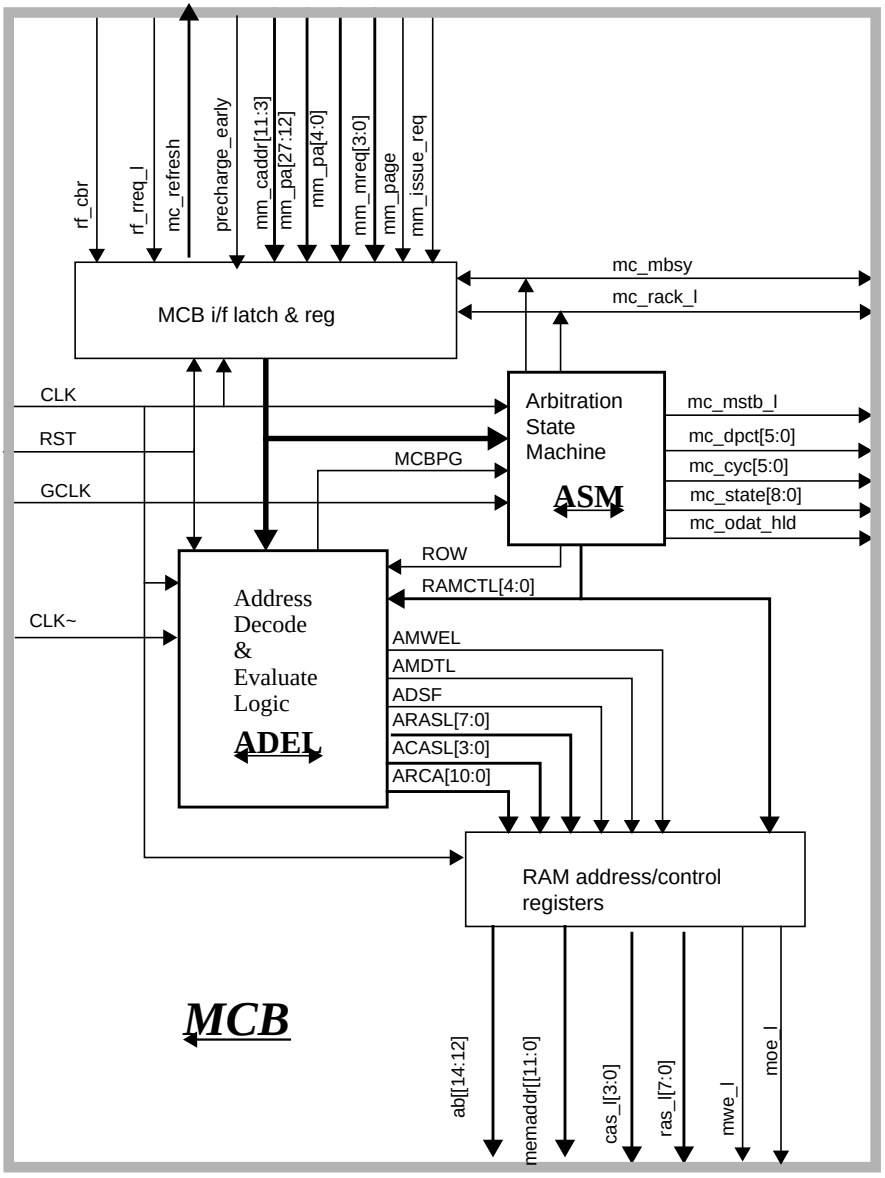


Figure 8-1 Memory Control Block diagram

8.3.1 Arbitration State Machine (ASM)

The ASM detects the requests from the MMU and Refresh blocks, arbitrates between them if necessary, and grants the appropriate request. Once a request is granted, the MCB carries out the requested memory operation of one or more memory cycles. *Table 8-3* lists all the types of memory operations performed by the MCB, the possible request sources and the type and number of cycles involved.

Table 8-3 Memory operations performed by MCB

Operation	Source	Memory Cycles produced
d.rd.32b	MMU. Used to fill one line of I-cache (Inst-Fetch).	32 bytes are read from DRAM in a single operation, using 4 longword (64bit) read cycles. The first read is paged or non-paged, from the address given on PA. The following 3 reads are paged. ADEL will supply the address for the next 3 reads, incrementing or wrapping it as necessary, in order to read a 32 byte aligned block and fill a whole I-cache line.
d.rd.16b	MMU. Used to fill one line of D-cache.	16 bytes are read from DRAM in a single operation, using 2 longword (64bit) read cycles. First read is a paged or non-paged cycle, using the address supplied on PA. The next cycle is a paged read, where ADEL will increment or wrap the address in order to read a 16byte aligned block from memory.
d.rd.8b	MMU. Used for IU longword reads.	8 bytes are read from DRAM, using a paged or non-paged longword read from the address supplied by PA.
d.wr.8b	MMU. Used for IU longword writes.	8 bytes are written to DRAM, using a paged or non-paged longword write to the address supplied by PA.
d.wr.4b	MMU. Used for IU word writes.	4 bytes are written to DRAM, using a paged or non-paged word write to the address supplied by PA.
d.rmw.2b	MMU. Used for IU byte writes.	a halfword (16bit) write to DRAM in a single operation, using a paged or non-paged word read followed by a paged word write, using the same address supplied by PA. MCB will perform the read and write cycles and will instruct DPC to latch the 16bit write-data from the source, insert it in the appropriate halfword of the word read from memory and then gate it back on memory data-bus as the write data.

Table 8-3 Memory operations performed by MCB *(Continued)*

Operation	Source	Memory Cycles produced
d.rmw.b	MMU. Used for IU byte writes.	a byte (8bit) write to DRAM in a single operation, using a paged or non-paged word read followed by a paged word write, using the same address supplied by PA. MCB will perform the read and write cycles and will instruct DPC to latch the 8bit write-data from the source, insert it in the appropriate byte of the word read from memory and then gate it back on memory data-bus as the write data.
cbr.ref	RFR. Used to do a refresh cycle on all DRAM	Will force a Cas-before-Ras refresh cycle to be performed on all DRAM banks with four banks refreshing at the same time.
d.wr.16b	MMU. Unused.	16 bytes are written to DRAM in a single operation, using 2 longword (64bit) write cycles. First write is a paged or non-paged cycle, using the address supplied by PA. The next cycle is a paged write, where ADEL will increment.

8.3.2 Arbitration for Memory Access and ASM Priority Scheme

All requests are checked at the end of each operation. For multi cycle operations, the checking is done at the end of the last memory cycle. The MEMIF arbitration scheme is based on the following rules.

- If no requests are pending, the MEMIF enters the idle state and remains there until a request is detected. If only one request is pending, it is granted and the operation begins. If more than one request is pending, the one with the highest priority is granted and the operation begins. The priorities are given as follows.
  - The MMU has the highest priority, except when the current cycle is also an MMU request, in which case it is considered the lowest priority. This is to prevent bus locking as a result of back to back MMU requests.
  - The PCIC has the second highest priority except when the current and last cycles are also PCIC requests.
  - A DRAM refresh request has the lowest priority, except when the current cycle is an MMU request, in which case it has a higher priority.
- If a DRAM refresh request is detected while MEMIF is in idle, the state machine advances to a check state, where it checks to see if an MMU request occurred just as the DRAM refresh request was accepted. If there are no pending MMU requests, MEMIF continues to acknowledge the DRAM refresh request and perform a DRAM refresh. Otherwise, it services the MMU cycle.

8.3.3 Address Decode & Evaluate Logic (ADEL)

This block primarily monitors the address and function-select signals coming from the MMU and RFR and performs the necessary decode and re-mapping of the memory address and control lines. Based on commands received from ASM, ADEL gates the low address (bits [4:3], and uses the result with the rest of the address bits driven from MMU) and with memory control signals required for the current memory operation.

The mapping of system memory is discussed in *Address Mapping For System DRAM* on page 128.

8.4 Data Alignment and Parity Check / Generate Logic (DPC)

During any read, write or hardware controlled read-modify-write cycle, DPC performs the necessary data alignment and byte / halfword placement. It also provides temporary storage for hardware-controlled read-modify-write cycles, resulting from byte / halfword write cycles to memory.

The DPC also contains the parity generation and checking logic. The parity comprises one bit per 32-bit word and is used for system DRAM only.

The type of parity operation for the system DRAM is determined by the state of the parity control bit (PC) and the parity enable control bit (PE) in the MMU processor control register. See Section 5.7.1, *Processor Control Register* on page 73 for the details.

Table 8-4 Parity Control Definition

PC	Description
0	Check / Generate even Parity
1	Check / Generate odd Parity

Since system parity is one bit per word, any byte or halfword store operation results in a hardware-controlled read-modify-write cycle. During the read part of such operation, the word parity is checked and if an error is detected, a parity error is generated. After the word has been updated to contain the new byte / halfword, a write operation is performed, which also updates the parity. MEMPAR[0] is associated with MEMDATA[31:0] while MEMPAR[1] is associated with MEMDATA[63:32].

The data flow and type of operations performed by the DPC are governed by the commands it receives from the MCB.

The DPC block diagram of *Figure 8-2*, shows the basic data paths connecting the 64-bit external memory bus (b_memdata[63:00]) to the 32-bit internal one (misc[31:00]). The parity check/generation logic is shown to be on the output path, but for input data, parity is checked after it is clocked into the registers and gated through the alignment multiplexer.

The alignment multiplexer is also used to combine and produce the output data during a read modify write sequence. The complexity of this multiplexer is reduced by having the byte or forward data which is to be written to memory, already in the correct position. This is done by the block sourcing the data on the misc bus (D or I cache, or IU).

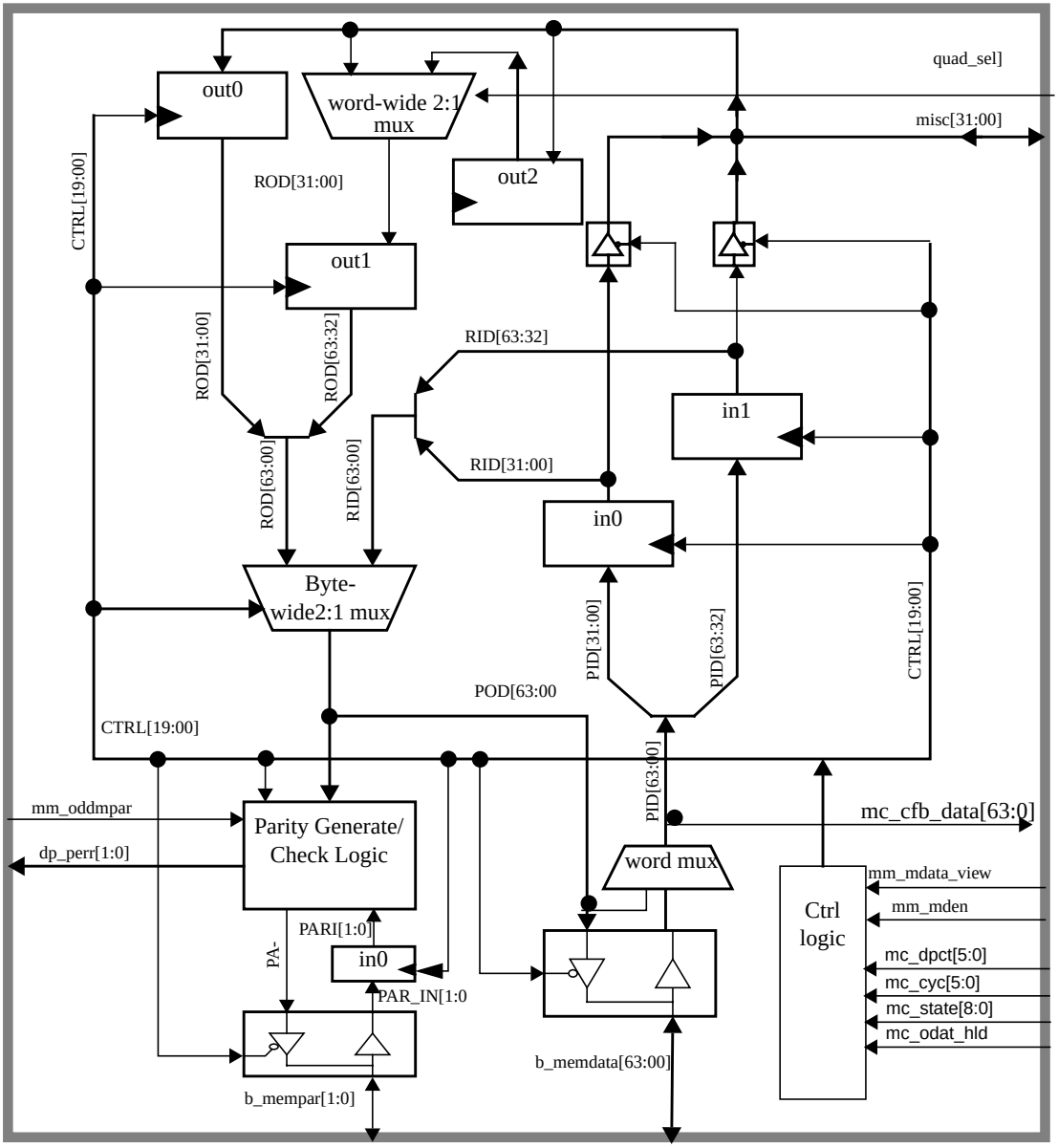


Figure 8-2 DPC Datapath and Parity Control Block Diagram

8.4.1 RAM Refresh Control (RFR)

The refresh control logic (RFR) is a simple request generator, asserting a request to MCB at fixed intervals. MCB will service this low priority request by performing a CAS-before-RAS type refresh cycle on all system RAM. Banks 0, 2, 4, 6 and banks 1, 3, 5, 7 will have RAS's asserted at a different cycle to reduce the magnitude of current spikes.

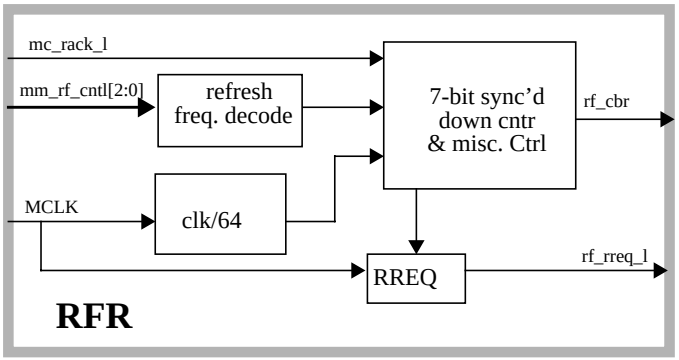


Figure 8-3 RAM Refresh Control block diagram.

RFR refresh rate can be selected by programming three bits of the Processor Control register according to the following table. These bits are then passed to RFR as mm_rf_cntl[2:0] input bits, which controls the rf_rreq_l rate.

Table 8-5 Refresh Rate Control bits.

mm_rf_cntl [2:0]	Refresh interval
0 0 0	Assert a refresh request once every 128 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 8.6MHz. This is the default after power up.
0 0 1	No Refresh!
0 1 0	Assert a refresh request once every 704 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 48MHz.
0 1 1	Assert a refresh request once every 896 CLK periods. With this setting, adequate refresh is guaranteed for CLK values of down to 60MHz.
1 0 0	Refresh every 1216 CLK periods to run above 83 MHz.

Table 8-5 Refresh Rate Control bits.

mm_rf_cntl [2:0]	Refresh interval
1 0 1	Refresh every 5120 clocks for low refresh DRAMs.
1 1 0	Refresh every 1408 CLK periods to run above 100 MHz.
1 1 1	Refresh every 1792 CLK periods to run above 125 MHz.

The RFR is also responsible for initializing the DRAMs on power-up.

After power-up and before they can be reliably used, DRAMs require a 500 µs *wait* period followed by eight CAS-before-RAS refresh cycles.

For systems built around the microSPARC-IIep CPU, the reset must remain active for at least 500us after power-up, to satisfy the *Wait* period. However, PCI subsystems require the source of the PCI reset signal to be stable 1.0ms after power has stabilized and.1 ms after clocks have stabilized. microSPARC-IIep systems should guarantee an active reset duration of 1.1ms or more.

After an active reset, the “mm_rf_cntl[2:0]” bits which reside in the MMU’s PCR register are set to “000” (See *Table 8-5*), setting RFR to generate a refresh request every 128 clocks. In addition, RFR itself, asserts its “rf_cbr” and “rf_rreq_l” signals, forcing MCB to enter a “cbr” state, where it performs eight CAS-before-RAS refresh cycles, completing the DRAM initialization cycle. After that, RFR negates both “rf_cbr” and “rf_rreq_l” signals, allowing the MCB to proceed to its normal operation state.

8.5 Clock Speeds

The microSPARC-IIep memory controller is designed to operate over a variety of clock frequencies, selected by means of speed select pins SP_SEL[2:0]. *Table 8-6* lists the four speeds available and the corresponding settings of SP_SEL[2:0].

Table 8-6 Processor Core Clock Speeds Available

sp_sel[2:0]	Clock Value	Comment
000	70 MHz	Low speed
001	85 MHz	Medium/low speed
010	100 MHz	Normal speed
011	133 MHz	High speed
100-111	–	Reserved for higher speeds

Wait states are inserted for medium speed compared to low speed; and higher speeds use even more wait states. For example, low speed has a read bandwidth of four cycles; medium speed, high speed and ultra high speed have 5, 6, and 7-cycle read bandwidths, respectively. The microSPARC-IIep processor timing is designed for systems that use 60 ns DRAM. Wait states and therefore the number of cycles of read bandwidth must be increased at higher clock speeds to maintain sufficient access-time margin for a given DRAM specification.

8.6 Summary of Cycles

Table 8-7 provides a summary of the number of cycles designed for different interface signals to the DRAM at various speed selects. Only cycles that are important to system usage are given here. This information provides the system designer with a quick reference to evaluate which types of DRAMs may be suitable for the speed select choice. Note that cycle numbers are given in terms of processor clock and not PCI clock. Actual delays from clock to output of each pin may differ.

Table 8-7 Number of Cycles for Different Interfaces

Parameter	Specification (ns)	Number of cycles at SP_SEL = 000	Number of cycles at SP_SEL = 001	Number of cycles at SP_SEL = 010	Number of cycles at SP_SEL = 011
t_RP	40	3.5	3.5	4.5	5.5
t_RAS (rd)	60	7.5	8.5	9.5	11.5
t_RAS (wr)	60	5.5	8.5	8.5	9.5
t_CP (rd)	10	1	1	2	2
t_CP (wr)	10	2	3	3	3
t_CAS (rd)	15	3	4	4	5
t_CAS (wr)	15	2	3	3	3
t_ASC	4	1	3	3	4
t_RAD, t_RAH	15-25, 10	1.5	1.5	1.5	1.5
t_RCD (rd)	20-40	3.5	3.5	4.5	5.5
t_RCD (wr)	20-40	2.5	4.5	4.5	5.5
t_DS, t_WCS	0, 4	1	3 - 2	3 - 2	4 - 2
t_DH, t_WCH	20, 19	2	3	3	3
t_RPC (ref)	10	2	2	2	2
t_CSR (ref)	15	1.5	1.5	2.5	3.5

Table 8-7 Number of Cycles for Different Interfaces

Parameter	Specification (ns)	Number of cycles at SP_SEL = 000	Number of cycles at SP_SEL = 001	Number of cycles at SP_SEL = 010	Number of cycles at SP_SEL = 011
t_CHR (ref)	20	4.5	4.5	4.5	6.5
t_RAS (ref)	60	6.5	6.5	6.5	8.5
t_RAS (rmw)	111	13.5	15.5	17.5	18.5
t_CAS1 (rd) (rmw)		3	4	4	5
t_CAS2 (wr) (rmw)		2	3	3	3
t_CP (rmw)		4	5	6	7

8.7 Memory Configurations

Memory configurations are illustrated in *Figure 8-4*, *Figure 8-5*, and *Figure 8-6*.

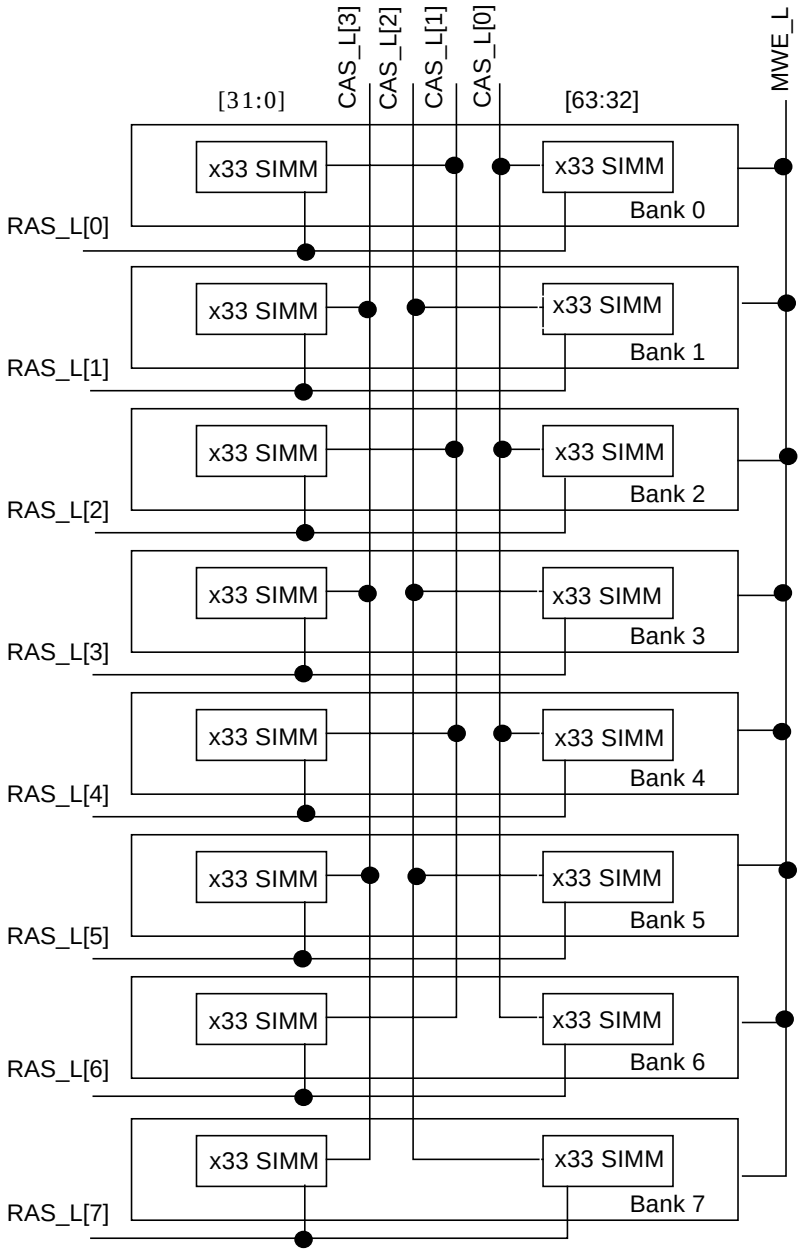


Figure 8-4 Dual-RAS Mode: Fast-Page Mode, 16-MB SIMMs (SIMM32_SEL=0)

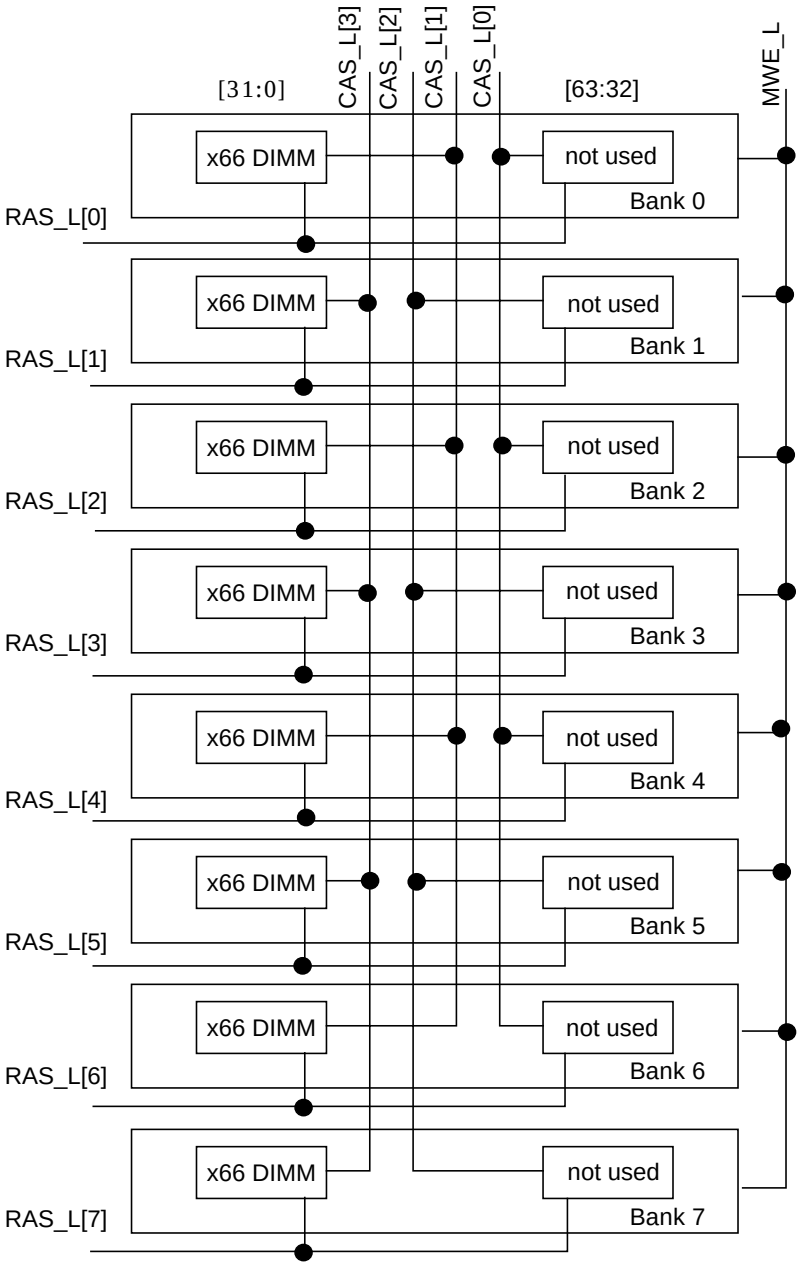


Figure 8-5 Single-RAS Mode: Fast-Page Mode, 32 MB SIMMs (SIMM32_SEL=1)

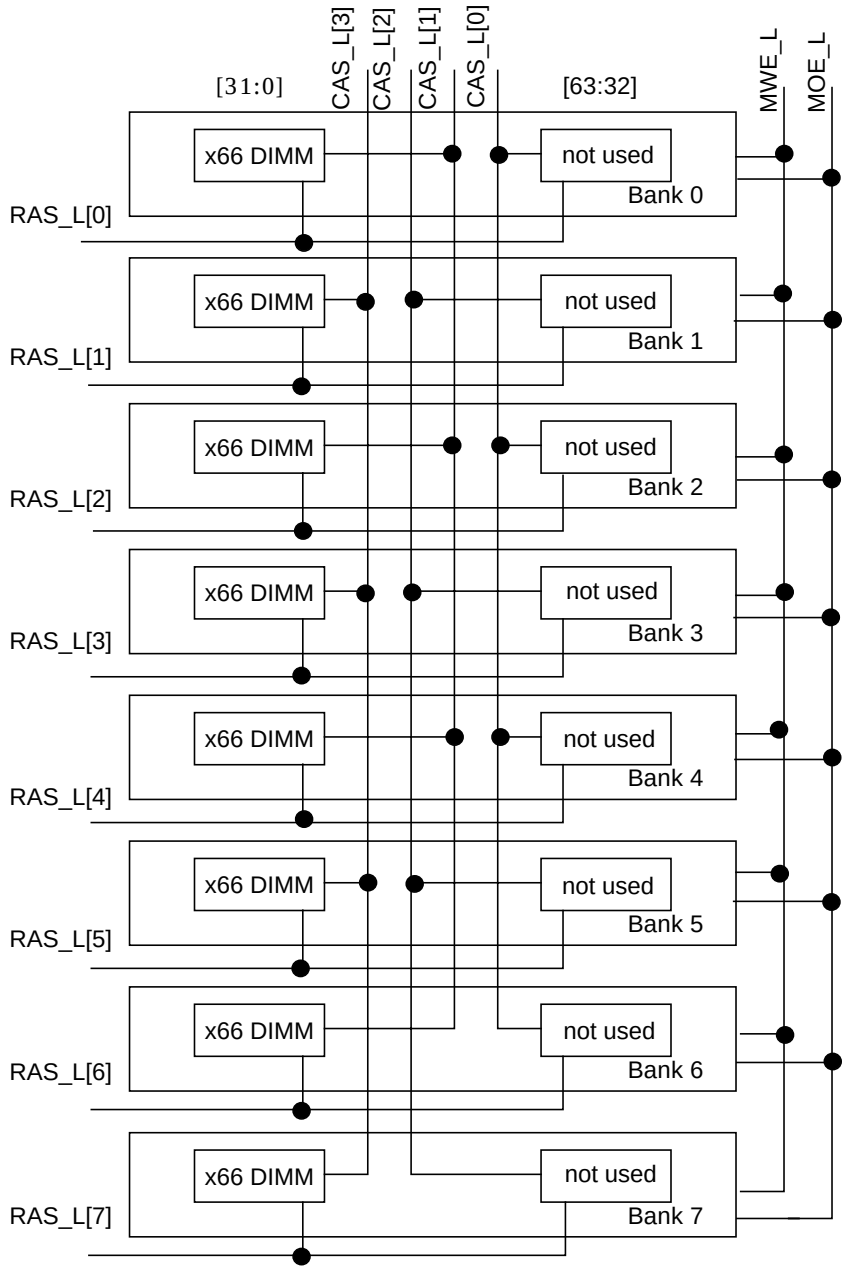


Figure 8-6 Single-RAS Mode: EDO, 32 MB DIMMs (SIMM32_SEL=1)

8.8 Local Bus (IAFX bus to PCIC) interface

An internal only version of a Local Bus is used within the microSPARC-IIep to interface to the PCI Controller. This internal version of the Local Bus is referred to as the IAFX bus. In addition to the same transfer characteristics as the AFX bus that were available in the microSPARC-II, the IAFX bus has the capability to allow another master access to the IAFX bus to initiate transfers. The following paragraphs explain how the PCIC accesses memory on behalf of PCI devices (DVMA).

The IAFX bus provides the interface from the PCIC to the microSPARC-IIep CPU. It uses an internal version of the data and address bus that connects to DRAMs. The microsparc-IIep allocates 256 megabytes of address space for the IAFX local bus (PCIC) physical address access. The PCIC responds to these addresses. This space is addressed by PA[30:28] being set to 011. The clock GCLK controls all transfers on the IAFX bus and is obtained by dividing the processor's clock by three. All internal local bus signals have timing requirements with respect to GCLK. There are a page-mode, and a non-page mode operation types depending on which address lines are changed from the previous access to the local bus space. There is no parity checking or generation associated with Local (IAFX) Bus access. (PCIC DVMA activity does generate parity when the IAFX bus is connected to the memory bus for DVMA writes to DRAM.)

The PCIC interface that connects to the IAFX bus has a four double-word deep FIFO for store (PIO) buffering. To avoid potential time-out problem, microSPARC-IIep allows 2.0 µs maximum operational latency for any Local Bus (IAFX) instructions. Local Bus interfaces, such as that of the PCIC and any PCI devices that reside on the PCI bus, should have a worst case latency of 2.0 µs, and an average latency of no more than 1.0 µs. If the 2.0 µs worst case reply latency is violated, the integrity of DRAM contents cannot be guaranteed.

To summarize the Local Bus (IAFX) interface features:

- Non-cached direct processor access - including bytes, halfword, word and double word access.
- Cached direct processor access.
- Full DMA access - supports sizes up to 32 bytes that microSPARC-IIep supports. All non-burst mode DMA operations are required functionality. Burst mode operations are consistent with existing design.
- Page mode detection to Local Bus (IAFX and PCIC) access.
- Suppressed parity checking.
- Hold off on processor read access until slave's write FIFO has zero or one writes pending. Also hold off on processor write access until a slot is available in the slave's FIFO. See Section 5.8.14, *Local Bus (PCIC Interface) Queue Level Register* and Section 5.8.15, *Local Bus (PCIC Interface) Queue Status Register*.

- Time-out declared when no response from Local Bus slave (PCIC) for 2047 GCLKs after the last request to Local Bus (PIO) is issued. This condition generates a Level 15 interrupt.
- Support bandwidth of one GCLK cycle (3 CPU cycles) per write access; and two GCLKs per read access.

CHAPTER 9

PCI Controller

9.1 Overview

The PCI Controller (PCIC) provides a bridge function between the 64-bit, internal, IAFX local bus and a 32-bit PCI bus. The high-bandwidth IAFX local bus can be connected to the main MicroSPARC-IIep processor system DRAM. Transfers on the IAFX local bus do not activate any external pins except when IAFX transfers to or from the main DRAM are undertaken. In this case the signals to or from the DRAMs are activated.

9.1.1 Features

The PCIC has the following key features.

- Complete 32-bit PCI interface
- Operation as host processor or intelligent satellite processor—see *Figure 9-1*
- Programmed Input/Output (PIO) transactions between the CPU and PCI slave devices
- Up to four Master/Slave external 32-bit PCI subsystems in host mode
- Direct Memory Access (DMA) transactions between PCI masters and host system memory
- 16-entry TLB provides address mapping from 32-bit PCI addresses to the 28-bit DRAM physical addresses.
- Direct transactions between PCI masters and PCI slaves.
- Selectable clock speed: CPU clock frequency is a multiple of input PCI clock frequency and pin-setting selectable

- The PCI interrupt controller supports up to eight external interrupts (bidirectional I/O pins) and generates interrupt vectors to the CPU. This controller can be disabled by the user if external interrupt vector generation is required
- Programmable configuration registers, always accessible from the CPU and accessible from the PCI bus when the microSPARC-IIep processor is configured in satellite mode
- On-chip PCI arbiter (which can be disabled) supports four external masters
- FIFO rate matching buffers between IAFX and PCI.
- Two 32 bit counters or one 32 bit counter and one 64 bit timer.

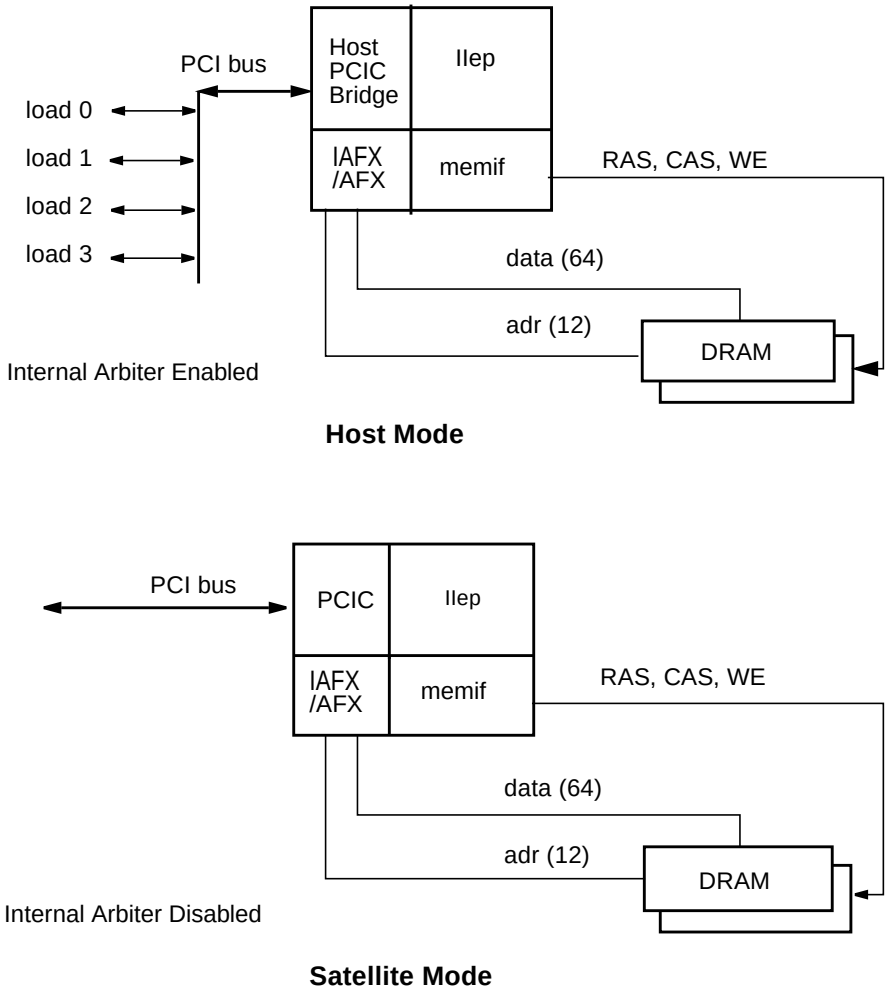


Figure 9-1 Host and Satellite microSPARC-IIep Modes

9.2 Data Translation (Endian Modes)

9.2.1 Overview

The PCIC interface to the processor takes place across an internal version of the IAFX bus. Refer to the AFX (Local) bus specification for details of the this bus. The internal version of the AFX bus is referred to as the IAFX bus in this document. Transfers across this IAFX bus do not activate external pins but the signals used and their timing are similar to those applicable for external AFX bus operation.

The first consideration is that the PCIC uses *little-endian* bit format, where bit 31 is the most significant bit, while SPARC uses a *big-endian* bit format, where bit 0 is the most significant bit. Consequently, the PCI address bit AD 0 equates to SPARC address bit 31, and the PCI address bit 31 equates to SPARC address bit 0.

The second consideration involves the byte ordering within the data that are comprised of more than a single byte. For little-endian data the least significant byte is stored at the lowest, or starting, address while the most significant byte is stored at the highest, or ending, address. For big-endian data, the most significant byte is stored at the lowest, or starting, address while the least significant byte is stored at the highest, or ending, address. The PCIC translates the data and address internally, with the IAFX interface numbered according to big-endian format. This allows for straightforward system interconnect. The PCIC twists all data to little-endian format by reordering the bytes as they are brought into the PCIC. Therefore each of the PCIC internal FIFOs contain little-endian formatted address and data.

Refer to Section 1.4 for more information on endian support and operation.

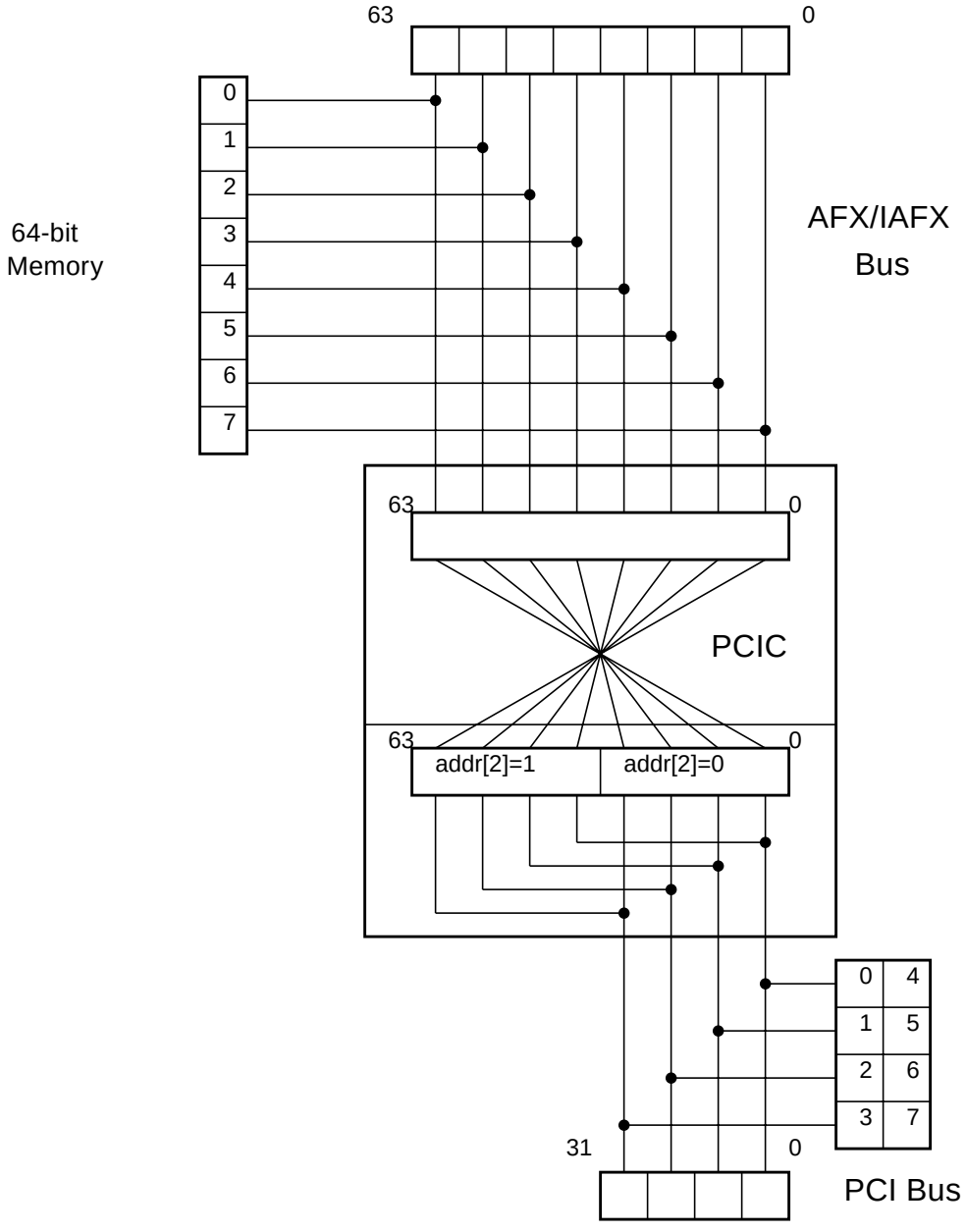


Figure 9-2 PCIC Byte Twisting

9.3 Memory Map and Address Translation

The PCIC memory map is defined within microSPARC-IIep as a 256-megabyte physical address space, in the range 0x3000.0000–0x3FFF.FFFF. This region maps all accesses to the PCI bus. The PCIC partitions this address space to memory map the different PCI cycle types supported. *Table 9-1* shows the fixed memory spaces for Type 0 and Type 1 PCI configuration cycles (with indexed access through address and data registers), PCI Special cycles, and a 64 kilobyte I/O cycle region that uses standard ISA I/O addresses. The PCIC supports two programmable regions for mapping the 28-bit IAFX physical address to the 32-bit PCI memory cycle physical addresses. There is also one programmable region supported for PCI I/O cycle types. The programmable memory map is discussed in the IAFX to PCI memory map section.

Table 9-1 microSPARC-IIep Memory Map

PA[30:28]	Address Space (256 MB partitions)
000	Main Memory (DRAM)
001	Control Space
010	Flash PROM (boot space available)
011	PCIC Bridge (boot space available)
100	Reserved Space
101	Reserved Space
110	Reserved Space
111	Reserved Space

The reset, and power-on, default memory map supports the fixed region decodes, and allocates the remainder of the 256-megabyte IAFX address region to PCI memory cycles (with no address translation, that is, AD[31:28] == 0). The fixed portion of the PCIC memory map represents only one megabyte of the address space that is reserved. The fixed I/O space region maps to the first 64 kilobytes of the PCI I/O cycle region. This may be overlapped with the programmable PCI I/O space region—see Section 9.3.1.

The mapping of PCI virtual addresses to microSPARC-IIep DRAM address is described in the subsequent sections. This mapping is performed by the IOTLB, which is managed by software. This is described in section 9.3.2.

Note – The capability of booting executable code through the PCI bus is allowed when the boot mode select pins are set to select booting from the PCI bus. In this case, there must be a subsystem on the PCI bus that responds to the memory read request immediately after reset, without configuration operations as are generated by the PCIC bus master. Refer to Section 11.7 for a table of boot mode addresses.

9.3.1 IAFX to PCI Memory Map

The PCI memory address space and the PCI I/O address space are mapped into the upper 255 megabytes of the PCIC physical address space. The map of PCI memory and PCI I/O address spaces are controlled by a set of programmable registers, known as translation registers. The translation registers provide software control for mapping the PCI address spaces of the PCI bridge anywhere in the physical address space. These registers work in concert with a fixed memory map for control, I/O and configuration space specified in the first one megabyte. That is, these registers define the *expansion range* of the specific memory maps. Although they may be programmed to overlap control spaces, the control space definitions of the memory map selected have precedence. The register set that accomplishes the mapping has separate control for PCI memory and PCI I/O spaces. *Table 9-2* illustrates possible address space mappings.

Table 9-2 PCIC Fixed Memory Map

	AFX (CPU) Physical Address[2]			
PCI cycle	**Byte 0**	**Byte 1**	**Byte 2**	**Byte 3**
I/O cycle (64 kB)	0011 0000	0000 0*xxx*	aaaa aaaa	aaaa aaaa
Configuration Address	0011 0000	0000 100*x*	xxxx xxxx	xxxx xxxx
Configuration Data[1]	0011 0000	0000 101*x*	xxxx xxxx	xxxx x***
PCIC Registers	0011 0000	0000 110*x*	xxxx xxxx	aaaa aaaa
Special cycle	0011 0000	0000 1110	xxxx xxxx	xxxx xxxx
Interrupt Acknowledge	0011 0000	0000 111*1*	xxxx xxxx	xxxx xxxx
Pass-through memory (without SIMBARs)	0011 0000	(!= 0000) *aaaa*	aaaa aaaa	aaaa aaaa

1. The three least significant bits of the physical address used to access the configuration data space must match address specified by the configuration address register
2. Symbol key: *a* is a bit in an address; *x* is a bit whose value is ignored

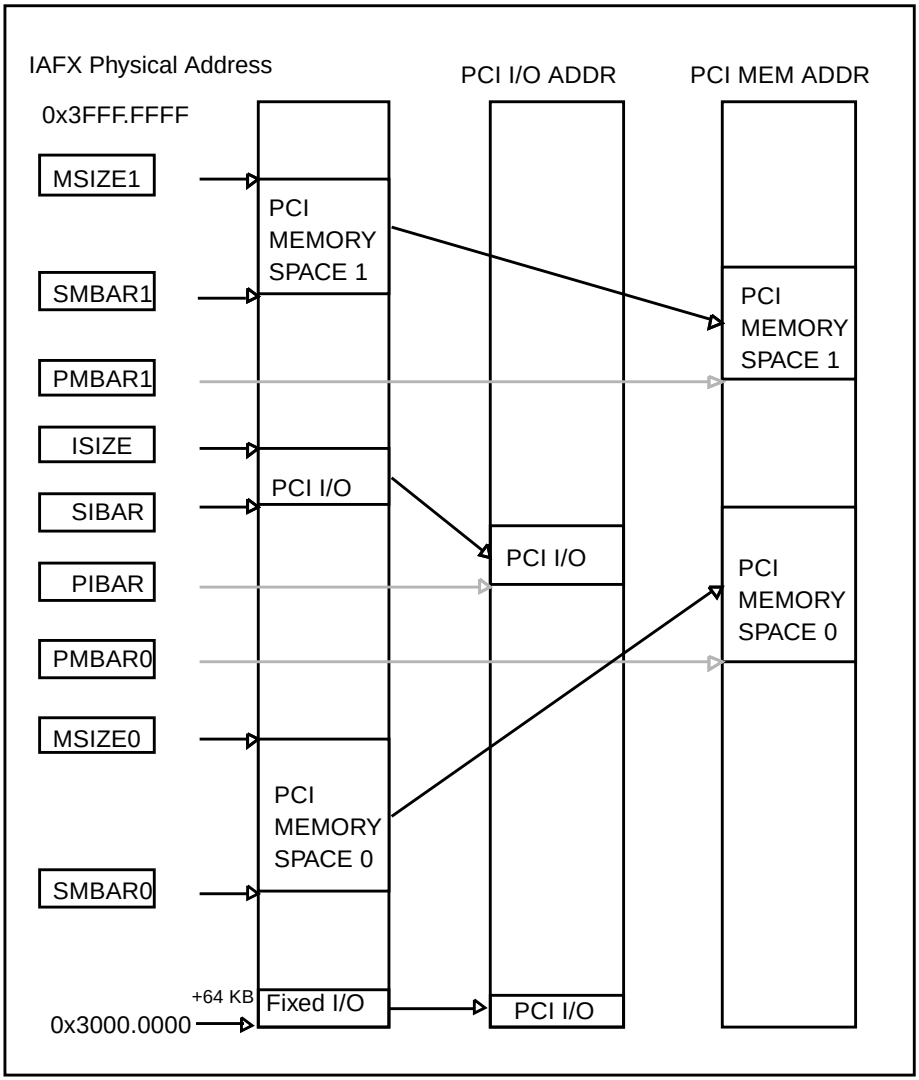


Figure 9-3 IAFX to PCI Addressing

Figure 9-3 shows how the PCIC bridge might map the IAFX bus memory address to the PCI bus. The PCIC monitors the address on the IAFX bus, and translates those operations decoded to fall within the range specified by the SMBAR0/MSIZE0, SMBAR1/MSIZE1 or SIBAR/ISIZE registers. The programmable size for each field can be set to 32 megabytes, 64 megabytes, 128 megabytes, or the entire 256 megabytes. If the IAFX address falls within these ranges, the PCIC translates the address using PMBARn/PIBAR to generate the appropriate PCI operation. There is

a built-in priority decode of the address, which resolves an overlapping address space definition. The fixed address map has highest priority, followed by memory translation registers—with 0 having higher priority than 1—and the I/O translation register with the lowest priority. This priority decode can be used to define address maps with regions other than the sizes specifically defined. The following equations show how the PCIC derives the PCI address from the IAFX bus.

PCI address =

{(PMBARn[7:4]), ((IAFX PA[27:24] & ~MSIZEn[3:0]) | PMBARn[3:0]),(IAFX PA[23:00])};

In these formulae, the 4-bit size field is used as a mask to remove the affected IAFX address bits and allow the PCI address bits to be merged. The PMBARn and PIBAR registers always prefix the most significant nibble (Addr[31:28]) on the resulting PCI address. An address that is not translated is routed to the PCI bus untranslated.

Note – Any microSPARC-IIep processor-generated system memory or I/O transaction on the PCI bus should not use addresses within the range accepted by the PCIC slave. System memory operations on the PCI bus that are accepted by the PCI slave interface, result in a memory operation request back to the IAFX bus. This loopback condition is forbidden—see Section 9.3.2, *PCI to IAFX Memory Map*.

Table 9-3 PCIC PIO Address Decode Priority

Priority	Address space
1 (highest)	Fixed address map (First 16 MB)
2	SMBAR0
3	SMBAR1
4	SIBAR
5 (lowest)	Pass Through (256 MB)

9.3.2 PCI to IAFX Memory Map

The PCIC allows PCI memory or I/O cycles to access the system DRAM from any PCI master except the microSPARC-IIep host itself. The PCI memory and I/O address range acknowledged (DEVSELd) by the PCIC is based on the value loaded into one of the six PCI Base Address Registers (PCIBARn) and the value loaded in the PCI address space Size register (PCISIZEn). These registers define the range of virtual addresses that are accepted for memory or I/O transactions that are mapped into the microSPARC-IIep DRAM.

PCI memory or I/O cycles cannot access any of the other address spaces defined in the microSPARC-IIep memory map. These spaces must be reached by DMA operations through memory. The software implementation should ensure that a DMA access is within the range of populated memory because his task is not performed by the hardware.

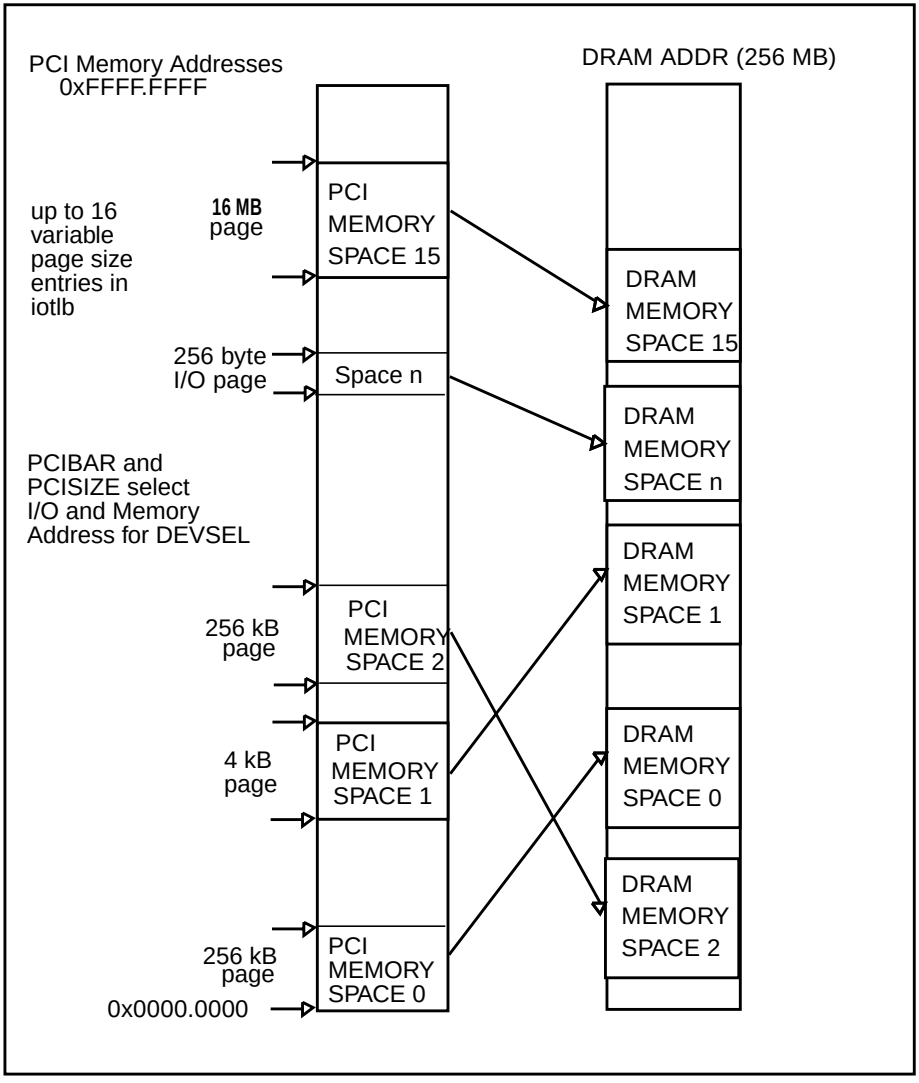


Figure 9-4 PCI to microSPARC-IIep DRAM mapping

The mapping from an accepted (DEVSELd) PCI slave memory transaction to the MicroSPARC-IIep DRAM memory is done by the IOTLB. The IOTLB provides a fully associative 16-entry PCI to DRAM address mapping. The IOTLB is managed by software with no table walking provided by hardware. All PCI mappings must be put into the IOTLB before the DMA operation is started. An accepted PCI memory address that does not match a translation entry has its address captured and an error interrupt signaled. The memory operation completes using a direct mapped (untranslated) address.

9.4 PCI Bus Interface

This section describes the microSPARC-IIep implementation of the PCI local bus. See the PCI Specification reference in *Bibliography* on page 275. *Table 9-4* lists the basic PCI bus operations and restrictions.

9.4.1 Basic PCI Bus Operations / Restrictions

Table 9-4 Basic PCI Bus Operations and Restrictions

Operation	Restriction
Addressing modes	Only the linear incrementing addressing mode is supported.
Master / slave modes	The microSPARC-IIep processor can either operate in Host Master mode or Satellite, or slave, mode. In Host Master mode, the PCIC may use the internal arbiter and generate PCI resets when appropriate. Also in Host master mode, the generation of configuration operations as a PCI master are supported. In Satellite mode, the internal arbiter is disabled and the PCIC is not allowed to drive PCI reset. In satellite mode, PCI configuration operations are supported. The req1_ pin is used as the idsel input pin. (The arbiter inputs are not required when operating in Satellite mode.) satellite mode is selected by having the pll_byp_l and ext_clk2 input pins both set to a 1 on power up. (The ext_clk1 pin can be connected to the system PCI clock source.) Host Master mode is selected when either of these two pins are at logic 0 on power up (The ext_clk1 pin can be connected to a 33 MHz clock source)

Table 9-4 Basic PCI Bus Operations and Restrictions *(Continued)*

Operation	Restriction
Configuration cycles	The PCIC can generate both type 0 and type 1 configuration accesses as a bus master. The technique of resistively coupling the drive of the IDSEL lines is used, as described in the PCI specification. The configuration registers that are contained within the PCIC are only accessible through PCI configuration cycles during PCI satellite mode.
Cache support	The PCIC does not support any cache operations.
Exclusive access	The PCIC does not implement locking at all and the LOCK# signal is not connected. Any exclusive access proceeds as if it were a non-exclusive access.

Before making the PCI bus quiescent, the microSPARC-IIep CPU ensures that there is no memory activity outstanding. This action ensures that all outstanding memory transactions are complete prior to the completion of a quiescent bus read. See section 9.5.4 (PCIC DVMA (IAFX Master) Control Register) for more details.

Note – When performing PCI configuration by accessing the address space of configuration address and configuration data, the three least significant address bits used for the configuration data space access must be identical to those for the previously loaded configuration address space access. For example, if the configuration address register is loaded with 0b100 in the three least significant bits, then the configuration data access must also contain 0b100 in the three least significant bits.

9.4.2 PCI Host / Satellite Mode

The microSPARC-IIep CPU can be programmed to operate in PCI host mode or satellite mode.

In PCI host master mode:

- The PCI arbiter is enabled and is responsible for asserting PCI reset.
- The microSPARC-IIep CPU is responsible for configuring other PCI entities with PCI configuration transactions.

In PCI satellite mode:

- The PCI arbiter is disabled.
- External configuring of PCI registers via PCI configuration transactions is disallowed.

The microSPARC-IIep CPU operates in satellite mode if PLL_BYP_L and EXT_CLK2 input pins are both tied to 1 at power-up. Otherwise, it operates in host master mode.

The PCI mode is visible to and programmable by software. Refer to Section 9.9.1, *System Status and System Control (Reset) Register*

Table 9-5 PCIC Slave Accepted Commands

Command	C/BE	Accepted
Interrupt acknowledge	0000	No
Special cycle	0001	No
I/O read	0010	Yes
I/O write	0011	Yes
reserved	0100	No
reserved	0101	No
Memory Read	0110	Yes
Memory Write	0111	Yes
Reserved	1000	No
Reserved	1001	No
Configuration read	1010	Yes, only in satellite mode
Configuration write	1011	Yes, only in satellite mode
Memory read multiple	1100	Treated as Memory Read
Dual address cycle	1101	No
Memory read line	1110	Treated as memory read
Memory write & invalidate	1111	Treated as memory write

Table 9-6 lists the PCIC-generated commands.

Table 9-6 PCIC Master Generated Commands

Command	C/BE	Generated	Notes
Interrupt acknowledge	0000	yes	
Special cycle	0001	yes	
I/O read	0010	yes	Not to microSPARC-IIep CPU's own I/O space
I/O write	0011	yes	Not to microSPARC-IIep CPU's own I/O space

Table 9-6 PCIC Master Generated Commands *(Continued)*

Command	C/BE	Generated	Notes
Reserved	0100	no	
Reserved	0101	no	
Memory read	0110	yes	not to microSPARC-IIep CPU's own memory
Memory write	0111	yes	not to microSPARC-IIep CPU's own memory
Reserved	1000	no	
Reserved	1001	no	
Configuration read	1010	yes (type 0 and type1)	not to microSPARC-IIep CPU's own config. registers
Configuration write	1011	yes (type 0 and type1)	not to microSPARC-IIep CPU's own config. registers
Memory read multiple	1100	no	
Dual address cycle	1101	no	
Memory read line	1110	no	
Memory write & invalidate	1111	no	

9.5 PCIC Control

The PCIC control is accessed through a set of registers in the PCIC—see *Table 9-7*. These registers can be accessed through the microSPARC-IIep CPU through the PCIC configuration register space. Some of these registers are standard PCI configuration registers as defined by the PCI specification. Some of these registers control specific operations within the PCIC itself.

Table 9-7 Configuration / Control Register Addresses

Offset	Number of Bytes	Register Name	Details in Section
00	4	Device and vendor ID	9.5.2.1
04	2	PCI Command register	9.5.2.2
06	2	PCI Device Status	9.5.2.3

Table 9-7 Configuration / Control Register Addresses *(Continued)*

Offset	Number of Bytes	Register Name	Details in Section
08	1	Revision	9.5.2.1
09	3	Class Code	9.5.2.1
0C	1	Cache Line-Size	9.5.3
0D	1	Latency Timer	9.5.3
0E	1	Header Type	9.5.2.1
0F	1	BIST	9.5.3
10 / 14 / 18 / 1C / 20 / 24	4	PCI Base Address register (PCIBAR0 / 1 / 2 / 3 / 4 / 5)	9.5.5.1
40	4	PCI counters (Retry and Trdy)	9.5.3
68	2	PCI Discard Timer (Half word)	9.5.3
44 / 48 / 4C / 50 / 54 / 58	4	PCI address space Size (PCISIZE0 / 1 / 2 / 3 / 4 / 5)	9.5.5.1
60	1	PCIC PIO (IAFX Slave) Control	9.6.3
62	1	PCIC DVMA (IAFX Master) Control	9.6.4
63	1	PCIC Arbitration / Interrupt Control	9.6.5
64	4	PCIC Processor Interrupt Pending register	9.7.5
6A	2	PCIC Software Interrupt Clear register (Half Word)	9.7.6
6E	2	PCIC Software Interrupt Set register (Half Word)	9.7.6
70	4	PCIC System Interrupt Pending register	9.7.2
74	4	PCIC System Interrupt Target Mask register	9.7.4
78	4	PCIC System Interrupt Target Mask Clear register	9.7.4
7C	4	PCIC System Interrupt Target Mask Set register	9.7.4
83	1	PCIC Clear System Interrupt Pending register	9.7.3
88	2	PCIC Interrupt Assignment Select register	9.7.1
8A	2	PCI Arbitration Assignment Select register	9.6.1
8c	2	PCIC Intrerrupt Assignment Select register	9.7.1
8e	1	PCIC Hardware Interrupt Output Register	9.7.7
84	4	PCI IOTLB Control register	9.5.7.3
90	4	PCI IOTLB RAM Input register	9.5.7.1
94	4	PCI IOTLB CAM Input register	9.5.7.2
98	4	PCI IOTLB RAM Output register	9.5.8.1
9C	4	PCI IOTLB CAM Output register	9.5.8.2
A0	1	System Memory Base Address register (SMBAR0)	9.5.4.1
A1	1	Memory address space Size (MSIZE0)	9.5.4.1
A2	1	PCI Memory base Address register. (PMBAR0)	9.5.4.1

Table 9-7 Configuration / Control Register Addresses *(Continued)*

Offset	Number of Bytes	Register Name	Details in Section
A4	1	System Memory Base Address register (SMBAR1)	9.5.4.2
A5	1	Memory address space Size (MSIZE1)	9.5.4.2
A6	1	PCI Memory base Address register. (PMBAR1)	9.5.4.2
A8	1	System I / O Base Address register (SIBAR)	9.5.4.3
A9	1	Memory address space Size (IOSIZE)	9.5.4.3
AA	1	PCI Memory base Address register. (PIBAR)	9.5.4.3
AC	4	Processor Counter Limit register or User Timer MSW	9.8.2
B0	4	Processor Counter register or User Timer LSW	9.8.3
B4	4	Processor Counter Limit register (non-resetting port)	9.8.4
B8	4	System Limit register	9.8.5
BC	4	System Counter register	9.8.6
C0	4	System Limit register (non-resetting port)	9.8.7
C4	1	Processor Counter User Timer Start / Stop register	9.8.8
C5	1	Timer Configuration register	9.8.9
C6	1	Counter Interrupt Priority Assignment Level register	9.8.10
C7	1	PIO Error Command register	9.5.9
C8	4	PIO Error Address register	9.5.9
CC	4	PCIC IOTLB Error Address register	9.5.8.3
D0	1	System Status and Control register	9.1

9.5.1 Configuration Register Accessing

The PCIC configuration registers can always be accessed by the CPU. The PCIC registers can be accessed through the PCIC fixed space register map. PCIC maps the registers to the address space starting at 0x300C.00xx, where the least significant byte defines the register offset. The register offset in the PCIC fixed space register map is the same as it is for the PCI configuration Space Header. All PCIC registers are defined in Little Endian (LE) format. This is because the registers are defined to be consistent with the PCI Local Bus Specification, which defines all PCI devices as LE format devices. When programming the PCIC, registers that can be accessed as byte registers, can be accessed by the byte, thus eliminating any potential confusion with regards to “endian-ness”. However, PCIC registers can also be accessed as any size, up to and including a word access.

Note – The registers in the PCIC can be accessed as little-endian or big-endian. Refer to the DVMA (IAFX Master) control registers for a description of the selection on the mode. This allows the access method to be selectable by software, and registers that are used for little-endian control can be accessed as little-endian and registers, such as those that control the interrupt controller or IOTLB, can be accessed as big-endian representations. Refer to Section 1.3, *microSPARC-IIep Endian Support* on page 4 for a description of the endian support.

9.5.2 PCI Configuration Register Definitions

This section describes the function of the PCI configuration registers supported by the PCIC. These registers are defined in Little Endian (LE) format, as per the PCI Local Bus Specification. The register definitions are divided into sections according to their function.

9.5.2.1 PCI Device Identification

Five fields in the PCI configuration header define the device identification (see *Table 9-8* through *Table 9-11*). All PCI devices implement these fields, for standard software identification. Each register is read-only.

Table 9-8 PCI Vendor ID Register:4 bytes @ offset = 00

Bit(s)	Reset	Field name	R/W
31:16	0x9000	Device ID -- - 0x9000	R
15:00	0x108e	Vendor ID. Sun Microelectronics - 0x108e	R

Table 9-9 PCI Revision Register: 1 byte @ offset = 08

Bit(s)	Reset	Field name	R/W
07: 00	00	First revision of PCIC	R

Table 9-10 PCI Class Code Register: 3 bytes @ offset = 09

Bit(s)	Reset	Field name	R/W
23: 16	06	Base class code - bridge device	R
15: 08	00	Sub-class code - other bridge device	R
07: 00	00	Programming interface - not applicable	R

Table 9-11 PCI Header Type Register: 1 byte @ offset = 0E

Bit(s)	Reset	Field name	R/W
07: 00	00	Header type	R

9.5.2.2 PCI Device Control

The PCI Command Register (*Table 9-9*) provides coarse control over the PCIC's ability to generate and respond to PCI cycles. When a 0 is written to bits [02:00] of this register, the PCIC is logically disconnected from the PCI bus for all accesses.

Table 9-12 PCI Command Register: 2 bytes @ offset = 04

Bit(s)	Reset	Field name	R/W
15:10	0	reserved; read as zero	R
09	0	Fast Back-to-back Enable; read as zero	R
08	0	SERR# enable	R/W
07	0	Address Stepping; read as zero.	R
06	0	Parity Check Enable	R/W
05	0	VGA Palette Snooping; read as zero.	R
04	1	Memory Write And Invalidate; read as one (treated as a memory write)	R
03	0	Special Cycle Support; read as zero	R
02	1	PCI Bus Master[1]	R/W
01	0	Memory Space [1]	R/W
00	0	I/O space [1]	R/W

1. Should be set to 1 for normal operation

9.5.2.3 PCI Device Status

The PCI Status Register is used to record status information for PCI bus related events. Reads to this register behave normally. Writes to the PCI status register can reset individual bits, but can not set any bits. A bit is reset whenever the register is written, and the data in the corresponding bit location is a 1. For example, to clear the system Error bit[14] and not affect any other bits, write the value 0b0100.0000.0000.0000.

Table 9-13 PCI Status Register: 2 bytes @ offset = 06

Bit(s)	Reset	Field Name	R/W
15	0	Detected parity error	R/C
14	0	Signaled SERR#	R/C
13	0	received Master Abort	R/C
12	0	received Target Abort	R/C
11	0	signaled Target Abort	R/C
10: 09	00	DEVSEL# timing - Medium=01 Refer to Section 9.6.5, *PCIC Arbitration Control Register*	R
08	0	Data Parity Error Detected While a Master	R/C
07	0	Fast Back-to-back Capable; read as zero	R
06	1	User Definable Features; read as one	R
05	0	66 MHz capable; read as zero	R
04	0	Master Retry Count Expired Read or Clear Note: this bit is not standard in PCI	R/C
03	0	Master Trdy Count Expired Read or Clear Note: this bit is not standard in PCI	R/C
02: 00	0	reserved; read as zero	R

9.5.3 PCI Miscellaneous Functions

The following three registers must have a defined response for PCI configuration accesses. The PCIC does not implement or support cache coherence on the PCI bus, and therefore the PCI cache line size register is set to zero (not actually implemented). The PCIC also does not implement any type of built-in self test, and therefore the PCI BIST register is supported only with “read as zero”. The PCI latency timer register is implemented as recommended in the PCI 2.1 specification, as an 8-bit register with the bottom three bits read-only, resulting in a timer granularity of eight clocks. The PCI latency timer is used to determine how long PCIC, as a master, is allowed to burst on the PCI bus.

Table 9-14 PCI Cache Line Size Register: 1 byte @ offset = 0C

Bit(s)	Reset	Field name	R/W
07: 00	00	no support for PCI cache. Read as zero	R

Table 9-15 PCI Latency Timer Register: 1 byte @ offset = 0D

Bit(s)	Reset	Field name	R/W
07: 03	00000	PCI Latency Timer.	R / W
02: 00	000	reserved	R

Table 9-16 PCI BIST Register: 1 byte @ offset = 0F

Bit(s)	Reset	Field name	R/W
07: 00	00	PCI BIST; no BIST. read as zero	R

Table 9-17 PCI Counters: 4 bytes @ offset = 40

Bit(s)	Reset	Field name	R/W
31:24	00	unimplemented (reserved)	R
23:16	00	PCI Trdy Counter	R / W
15:08	00	PCI Retry Counter	R / W
07: 00	00	unimplemented (reserved)	R

Note – The PCI Trdy Counter and the PCI Retry Counter are for diagnostic testing only. Setting these counters to anything other then the default value of zero may result in a violation of PCI protocol, and should not be done.

Table 9-18 PCI Discard Counters: 2 bytes @ offset = 68

Bit(s)	Reset	Field name	R/W
15: 00	0x7f	Discard Timer	R

Note – The PCI Discard Counter is for diagnostic testing only. Setting these counters to anything other then the default value of 0x7f may result in a violation of PCI protocol, and should not be done. The discard timer is used to discard data that has been fetched and is pending transfer

9.5.4 Processor (IAFX) to PCI Translation Registers (PIO)

The PCIC translation registers are used to map the processor's 28-bit physical address received from the IAFX bus (PIO) into a 32-bit PCI physical address. The translation registers function as groups, with two groups for mapping PCI memory cycles and one group for mapping PCI I/O cycles. If the IAFX physical address matches one of the system translation registers in the group, the physical address is translated accordingly. If the IAFX physical address does not match any of the system translation registers, or is not in the first one megabyte of the PCIC address space (fixed memory map) then the address is passed directly to the PCI bus, untranslated.

9.5.4.1 PCI Memory Cycle Translation Register Set 0

The PCI Memory Cycle Translation Register Set 0 is comprised of three registers: SMBAR0, MSIZE0, and PMBAR0. These are detailed in *Table 9-19*, *Table 9-20*, and *Table 9-21* respectively. The 4-bit value stored in SMBAR0 is used to compare with the IAFX physical address (PA) [27:24]. Both the IAFX PA and the SMBAR0 values are first masked (ANDed) with the contents of MSIZE0.

Address Match 0 = ((IAFX PA[27:24] & MSIZE0[3:0]) = = (SMBAR0[3:0] & MSIZE0[3:0]));

If the result of the comparison is true, PMBAR0 is used to form the PCI memory cycle address according to this equation.

PCI address = {(PMBAR0[7:4]), ((IAFX PA[27:24] & ~MSIZE0[3:0]) | PMBAR0[3:0]),(IAFX PA[23:00])};

Note that the PCI memory address is always prefixed with PMBAR0[7:4], regardless of the size specified by MSIZE0. If the result of the address comparison is false, then no translation is performed based on PMBAR0.

Table 9-19 System Memory Base Address Register 0 (SMBAR0) (1 byte @ offset = A0)

Bit(s)	Reset	Field Name	R/W
07: 04	0b0000	reserved	R
03: 00	0b0000	System Memory Base Address [27:24]	R/W

Table 9-20 System Memory Size Register 0 (MSIZE0) (1 byte @ offset = A1)

Bit(s)	Reset	Field Name	R/W
07: 04	0x0	reserved	R
03: 00	0x0	System Memory Size	

mask for address bits[27:24]

Value	Memory Size	R/W
0xF	16 MB	R/W
0xE	32 MB	
0xC	64 MB	
0x8	128 MB	
0x0	256 MB	

Table 9-21 PCI Memory Base Address Register 0 (PMBAR0) (1 byte @ offset = A2)

Bit(s)	Reset	Field Name	R/W
07: 00	0x00	PCI Memory Base Address [31:24]	R/W

9.5.4.2 PCI Memory Cycle Translation Register Set 1

The PCI Memory Cycle Translation Register Set 1 is comprised of three registers: SMBAR1, MSIZE1 and PMBAR1. The 4-bit value stored in SMBAR1 is used to compare with the IAFX physical address (PA) [27:24]. Both the IAFX PA and the SMBAR1 values are first masked (ANDed) with the contents of MSIZE1.

Address Match 1= ((IAFX PA[27:24] & MSIZE1[3:0]) = = (SMBAR1[3:0] & MSIZE1[3:0]));

If the result of the comparison is true, PMBAR1 is used to form the PCI memory cycle address according to the equation:

PCI address = {(PMBAR1[7:4]), ((IAFX PA[27:24] & ~MSIZE1[3:0]) | PMBAR1[3:0]),(IAFX PA[23:00])};

Note that the PCI memory address is always prefixed with PMBAR1[7:4], regardless of the size specified by MSIZE1.If the result of the address comparison is false, then no translation is performed based on PMBAR1, MSIZE1, and PMBAR. See *Table 9-22*, *Table 9-23*, and *Table 9-24* respectively.

Table 9-22 System Memory Base Address Register 1 (SMBAR1) (1 byte @ offset = A4)

Bit(s)	Reset	Field Name	R/W
07: 04	0000	reserved	R
03: 00	0000	System Memory Base Address [27:24]	R/W

Table 9-23 System Memory Size Register 1 (MSIZE1) (1 byte @ offset = A5)

Bit(s)	Reset	Field Name	R/W
07: 04	0x0	reserved	R
03: 00	0x0	System Memory Size	

mask for address bits[27:24]

Value	Memory Size	R/W
0xF	16 MB	R/W
0xE	32 MB	
0xC	64 MB	
0x8	128 MB	
0x0	256 MB	

Table 9-24 PCI Memory Base Address Register 1 (PMBAR1) (1 byte @ offset = A6)

Bit(s)	Reset	Field Name	R/W
07: 00	0x00	PCI Memory Base Address [31:24]	R/W

9.5.4.3 PCI I/O Cycle Translation Register Set

The PCI I/O Cycle Translation Register Set is comprised of three registers: SIBAR, ISIZE and PIBAR. The 4-bit value stored in SIBAR is compared with the IAFX physical address (PA) [27:24]. Both the IAFX PA and the SIBAR values are first masked (ANDed) with the contents of ISIZE.

Address Match = ((IAFX PA[27:24] & ISIZE[3:0]) = = (SIBAR[3:0] & ISIZE[3:0]));

If the result of the comparison is true, PIBAR is used to form the PCI memory cycle address according to the equation:

PCI I/O address = {(PIBAR[7:4]), ((IAFX PA[27:24] & ~ISIZE[3:0]) | PIBAR[3:0]),(IAFX PA[23:00])};

Note that the PCI memory address is always prefixed with PIBAR[7:4], regardless of the size specified by ISIZE. If the result of the address comparison is false, then no translation is performed based on PIBAR.

Table 9-25 System I/O Base Address Register (SIBAR) (1 byte @ offset = A8)

Bit(s)	Reset	Field Name	R/W
07: 04	0x0	reserved	R
03: 00	0x0	System I/O Base Address [27:24]	R/W

Table 9-26 System I/O Size Register (ISIZE) (1 byte @ offset = A9)

Bit(s)	Reset	Field Name	R/W
07: 04	0x0	reserved	R
03: 00	0x0	System I/O Size (mask)	R/W

mask for address bits[27:24]

Value	Memory Size
0xF	16 MB
0xE	32 MB
0xC	64 MB
0x8	128 MB
0x0	256 MB

Table 9-27 PCI I/O Base Address Register (PIBAR) (1 byte @ offset = AA)

Bit(s)	Reset	Field Name	R/W
07: 00	00	PCI I/O Base Address [31:24]	R/W

9.5.5 PCI to DRAM (IAFX) Translation Registers and Operation

PCI transactions are accepted by the PCIC PCI slave interface, based on the transaction type (memory or I/O), and an acceptable address. The PCIC slave interface accepts memory or I/O transactions that match the address range specified in any one of the six PCI Base Address Registers. A full 32-bit address is presented

on the PCI bus and may be mapped into a 28-bit physical address to be used to access Main Memory (DRAM). The mapping from PCI addresses to DRAM addresses is done by the IOTLB.

9.5.5.1 PCI Base Address/Size Registers

The PCI base address registers contain the most significant 24 bits of the 32-bit base address for PCI operations that are accepted (DEVSELd) by the PCI slave interface for memory or I/O operations. The PCIC slave interface compares all memory and I/O requests presented on the PCI bus for this value. When the address presented on the PCI bus matches the value in the PCI Base Address register, along with the size specified by the PCI Size register, the PCIC slave accepts that memory or I/O operation—if enabled in the PCI Command register—and subsequently performs a memory operation on the DRAM. The address used to perform the main memory operation on the IAFX bus is subject to mapping using the PCI IOTLB if enabled (see next section). There are six base address registers and size registers sets.

bit 00, PCI I/O Base Address Select, selects between I/O addresses and memory addresses. When bit 00 is set to a one, the contents of this base address register—and the size register associated with it—are used to compare against I/O addresses that are received by the PCIC slave. When this bit is set to zero, the base address register and size register are used to compare against memory addresses received by the PCIC slave. These operations must also be enabled in the PCIC command register.

Table 9-28 PCI Base Address Registers (PCIBASE0: 4 bytes @ offsets = 10,14,18,1C,20,24)

Bit(s)	Reset	Field name	R/W
31:08	0	PCI Base Address Register [31:08]	R/W
07: 01	0	unused	R
00	0	PCI I/O Base Address Select	R/W

The PCI Memory Size (PCISIZE#) registers, shown in *Table 9-29*, are used to select the size of the address comparison. The bits that are written as ones allow the corresponding base address register bits to participate in the comparison. When a bit is set to a zero, the corresponding bit of the Base Address register is not used in the comparison of the PCIC-accepted address with the value in the base address register. The address bits that are not compared are still accepted and propagated to the IOTLB and DRAM. This allows I/O cycles that have been accepted within different 256-byte boundaries (mask set to all F's) to be mapped to the same page in the IOTLB.

Table 9-29 PCI Memory Size Register (PCISIZE0) (4 bytes @ offset = 44,48,4C,50,54,58)

Bit(s)	Reset	Field Name		R/W
31:08	0	System Memory or I/O Size		R/W
		mask for address bits[31:08]		
		Value	**Memory Size**	
		0xFF FF FF	256 B	
		0xFF FF FE	512 B	
		0xFF FF FC	1 kB	
		0xFF FF F8	2 kB	
		0xFF FF F0	4 kB	
		0xFF FF E0	8 kB	
		0xFF FF C0	16 kB	
		0xFF FF 80	32 kB	
		0xFF FF 00	64 kB	
		0xFF FE 00	128 kB	
		0xFF FC 00	256 kB	
		0xFF F8 00	512 kB	
		0xFF F0 00	1 MB	
		0xFF E0 00	2 MB	
		0xFF C0 00	4 MB	
		0xFF 80 00	8 MB	
		0xFF 00 00	16 MB	
		0xFE 00 00	32 MB	
		0xFC 00 00	64 MB	
		0xF8 00 00	128 MB	
		0xF0 00 00	256 MB	
		0xE0 00 00	512 MB	
		0xC0 00 00	1 GB	
		0x80 00 00	2 GB	
		0x00 00 00	4 GB	
07:00	0	unused		R

9.5.6 PCIC IOTLB Operation (DVMA)

Memory operations that are accepted by the PCIC slave interface (DVMA) are mapped to main memory DRAM addresses by the IOTLB. The IOTLB is a 16-entry fully-associative content-addressable memory (CAM) and random access memory (RAM) set that is fully managed by software. Five registers are provided in the PCIC to manage the IOTLB.

Before any read or write operations access the IOTLB, all pending PCI operations should be made quiescent. The control to ensure quiescence is provided in the PCI DVMA control register (configuration register 0x62). If a read or write to the IOTLB for control purposes is attempted at the same time as a normal PCI to DRAM access is tried, the translation attempt is aborted and an undefined address may be used to access the DRAM.

To ensure that the quiescent state is not extended any longer than necessary, interrupts may be disabled while the PCI bus is made quiescent.

The PCI IOTLB can contain three different -sized entries. Posted entries can match on 4-kilobyte page size, 256-kilobyte page size, or 16-megabyte page size. All three entry types can be resident in the IOTLB at the same time. Software should never allow multiple entries to be written into the IOTLB that can result in multiple matches. This rule also applies to mapping a 4-kilobyte or 256-kilobyte page inside another larger sized page.

The IOTLB can be flash flushed on any single page entry, or the entire IOTLB can be flushed at once.

The PCI IOTLB can be disabled by setting a bit in the PCI DVMA (IAFX Master) Control Register (configuration register 0x62). When the IOTLB is disabled, the addresses that have been accepted from the PCI slave for DRAM operations are untranslated and directly mapped into DRAM physical addresses. In this case the most significant PCI address bits [31:28] are ignored.

A block diagram of the IOTLB and associated registers is shown in *Figure 9-5*.

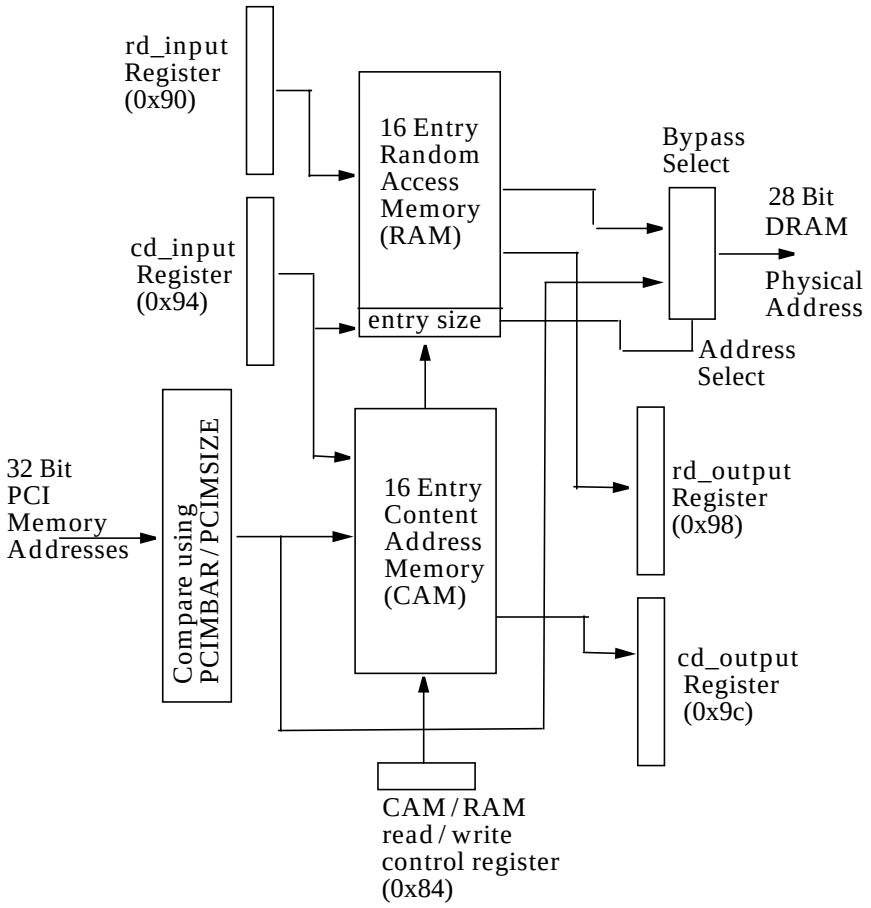


Figure 9-5 IOTLB Block Diagram with Control Registers

9.5.7 PCIC IOTLB Write Registers

The PCI IOTLB RAM Input register and the PCI IOTLB CAM Input register contain write data for the RAM and the CAM respectively. The PCI IOTLB control register is used to control the operation of the IOTLB. The IOTLB CAM and RAM are always accessed together. Normal content addressable searches result in a single match indicator, which is used to select the appropriate entry from the RAM.

A read operation is performed by selecting which entry number to read and writing an IOTLB read command into the IOTLB control register, then reading the CAM output register and the RAM output register.

A write operation is performed by first writing the CAM input register with the data to be written into the CAM (the PCI address to translate), and the RAM input register with the corresponding RAM data (the DRAM physical address). Following setting up of an entry for these two registers, the IOTLB control register is loaded with the entry number selected for the write, followed by the write command.

9.5.7.1 PCI IOTLB RAM Input Register

The PCI IOTLB RAM Input register (*Table 9-30*) contains physical address bits that are used to address the DRAMs as a result of a successful translation.

Table 9-30 PCI IOTLB RAM Input Register (PCIRIR) (4 bytes @ offset = 90)

Bit(s)	Reset	Field Name	R/W
31:28	0	unused (not written to RAM)	R/W
27:24	0	DRAM Physical Memory Address (4K/256K/16M)	R/W
23:18	0	DRAM Physical Memory Address (4K/256K)	R/W
17:12	0	DRAM Physical Memory Address (4K)	R/W
11:03	0	unused (written to ram also)	R/W
02:00	0	unused (NOT written to ram)	R/W

Bits 11:03 are written into the RAM, but do not participate in the translation process. These bits can be used to store entry information.

Bits 02:00 are unused and are not written into the RAM. The actual input to the RAM is the size of the translation being written to the CAM. The input to the RAM is derived from the input to the CAM—bit position[02:00]. The outputs from the RAM are used when the IOTLB is enabled and there is an IOTLB hit to select the portion of the real address to be overridden by the translated address. This allows multiple-sized entries to be placed in the CAM at once.

9.5.7.2 PCI IOTLB CAM Input Register

The PCI IOTLB CAM input register (see *Table 9-31*) contains address bits that are used to write information into the CAM for subsequent compares with PCI addresses.

Table 9-31 PCI IOTLB CAM InputReg.(PCICIR): 4 bytes @ offset = 94

Bit(s)	Reset	Field name	R/W
31:24	0	PCI Address to Translate (4 kB/256 kB/16 MB)	R/W
23:18	0	PCI Address to Translate (4 kB/256 kB)	R/W
17:12	0	PCI Address to Translate (4 kB)	R/W
11:04	0	unused (not written to CAM)	R
03	0	PCI Address Valid	R/W
02	0	PCI Address Check Enable for 16 MB Pages	R/W
01	0	PCI Address Check Enable for 256 kB Pages	R/W
00	0	PCI Address Check Enable for 4 kB Pages	R/W
Valid Combinations of bits 2:1:0 for compares		0b000 enables 4 kB page size for translation 0b001 enables 256 kB page size for translation 0b011 enable 16 MB page size for translation 0b111 disables all page size comparisons	
Valid Combinations of bits 2:1:0 for Flush Operations		000 flush entries that match on 4-kB page size 001 flush entries that match on 256-kB page size 011 flush entries that match on 16-MB page size 111 flush all entries	

Bit 03 is the valid bit, and must be set to a one for an entry to be valid. An entry must be marked valid in order to result in a successful translation. When bit 03 is set to a zero and written into the CAM, that entry is marked invalid and is not used for PCI address compares. The remaining portion of the CAM and RAM input registers are a *don't care* if the entry is to be written as invalid.

Bit 02 disables the comparison for 16-megabyte page sizes. When an entry is written into the CAM with bit 02 set to a zero, that entry is a 4-kilobyte, 256-kilobyte, or 16-megabyte page, and requires address bit 31:24 to match the IOTLB contents. The value of bit 02 is also written into position 02 of the RAM when the CAM is written.

Bit 01 disables the comparison for 256-kilobyte pages. When an entry is written into the CAM with bit 01 set to a zero, that entry is a 4 kilobyte or a 256-kilobyte page. This requires address bit 23:18 to match the IOTLB contents. The value of bit 01 is also written into the RAM in position 01 when the CAM is written.

Bit 00 disables the comparison for 4-kilobyte page sizes. When an entry is written into the CAM with bit 00 set to a zero, that entry is a 4-kilobyte page. This requires PCI bus address bit 17:12 to match the IOTLB contents. When bit 00 is set to a one, address bits 17:12 do not participate in the address comparison. The value of bit 00 is also written into the RAM in position 00 when the CAM is written.

When flushing the CAM, all valid bits that match the compare prior to the flush are set to zero after the flush completes.

9.5.7.3 PCI IOTLB Control Register

The PCI IOTLB control register—see *Table 9-32*—contains address bits used to compare with PCI addresses. When there is a match of the PCI address with the contents previously written into the CAM, a successful translation has been made.

Table 9-32 PCI IOTLB Control Register (PCICR) (1 byte @ offset = 84)

Bit(s)	Reset	Field Name	R/W
07	0	IOTLB Write Select	R/W
06	0	IOTLB Flush Enable	R/W
05	0	IOTLB Address Select	R/W
Valid Commands for bits 7:6:5			
	0b111	invalid; undefined	
	0b110	invalid; undefined	
	0b101	Directed Write of CAM and RAM at entry in “entry select” field	
	0b100	invalid; undefined	
	0b011	invalid; undefined	
	0b010	Flush (size of flush defined by bits 02:00 of CAM Input Register)	
	0b001	Directed Read of CAM and RAM at entry in “entry select” field	
	0b000	invalid; undefined	
Bit(s)	**Reset**	**Field Name**	**R/W**
04	0	unused	R/W
03:00	0	IOTLB Entry Select	R/W

9.5.8 PCIC IOTLB Read Registers

The PCI IOTLB RAM Output Register and the PCI IOTLB CAM Output Register are used to read data from the RAM and the CAM respectively. The PCI IOTLB Control Register is used to control the operation of the IOTLB and is described in Section 9.5.7.3, *PCI IOTLB Control Register*. The IOTLB CAM and RAM are always accessed together.

A directed read operation is performed by selecting which entry number to read and writing a IOTLB read command into the IOTLB control register. After writing the read command in the control register, the CAM output register and RAM output register may be read.

9.5.8.1 PCI IOTLB RAM Output Register

The PCI IOTLB RAM output register (see *Table 9-33*) contains physical address bits used to address the DRAMs as a result of a successful translation.

Table 9-33 PCI IOTLB RAM Output Register (PCIROR) (4 bytes @ offset = 98)

Bit(s)	Reset	Field Name	R/W
31:28	0	unused (read as zero)	R
27:24	0	DRAM Physical Memory Address (4K / 256K / 16M)	R
23:18	0	DRAM Physical Memory Address (4K / 256K)	R
17:12	0	DRAM Physical Memory Address (4K)	R
11:03	0	unused (read from RAM)	R
02:00	0	Page size selected, as written to CAM on input [02:00]	R

Bits 11:03 are read from the RAM, and can be used to store entry information, but do not participate in any translation process.

Bits 02:00 are read from the RAM and reflect the size of the translation entry. The input to the RAM is derived from the input to the CAM bit [02:00]. The outputs from the RAM are used when the IOTLB is enabled and there is a IOTLB hit to select the portion of the real address to be overridden by the translated address. This mechanism allows multiple-sized entries to be placed in the CAM at the same time.

9.5.8.2 PCI IOTLB CAM Output Register

The PCI IOTLB CAM output register (see *Table 9-34*) contains virtual address bits that have been written into the IOTLB. The IOTLB CAM output register is used to read entries previously written into the IOTLB. This facility is useful for diagnostic testing.

Table 9-34 PCI IOTLB CAM Output Register (PCICOR) (4 bytes @ offset = 9C)

Bit(s)	Reset	Field Name	R/W
31: 24	0	PCI Virtual Address to Translate (4 kB / 256 kB / 16 MB)	R
23:18	0	PCI Virtual Address to Translate (4 kB / 256 kB)	R
17:12	0	PCI Virtual Address to Translate (4 kB)	R
11:04	0	unused, read as zeros	R
03	0	Valid Bit as read from CAM	R
02:00	0	Page Size Selected, as read from CAM]	R

9.5.8.3 PCIC DVMA Error Address Register

The PCI DVMA Error Address Register captures a DVMA address that was used to translate (IOTLB) and then access memory. When an error is indicated, the address that was in use when the error occurred is captured in this register. For IOTLB access errors, the address captured is the address that was used in the unsuccessful IOTLB translation attempt. For DVMA parity errors, the address captured is the virtual address that was used to read from memory, plus eight. This error also generates a level-15 interrupt, and sets a bit in the interrupt registers to reflect this state.

Table 9-35 PCIC DVMA Error Address Register: 4 bytes @offset = CC

Bit(s)	Reset	Field name	R/W
31:03	0	PCI DVMA Address (or Address plus 8)	R
02: 01	0	Error Type code: 00: Translation failed in IOTLB access 01: Parity Error on DVMA read, wd0 10: Parity Error on DVMA read, wd1 11: Parity Error on DVMA read, both words	R
00	0	Access for IOTLB Operation Was a Read (=1)	R

9.5.9 PCIC PIO Error Command and Address Registers

The PCIC PIO error command and address registers reflect the command and address information that is current when an error is signaled during a PIO operation.

9.5.9.1 PCIC PIO Error Command Register

Table 9-36 PCIC PIO Error Cmd Register: 1 byte @offset = C7

Bit(s)	Reset	Field name	R/W
07: 04	0	reserved	R
03: 00	x	PCIC PIO Error Command	R

The PCIC PIO Error Address Register captures the address used in the last PIO command processed. For configuration Type 1 operations, this is not the same as the address that was presented on the PCI bus.

Table 9-37 PCIC PIO Error Address Register: 4byte @offset = c8

Bit(s)	Reset	Field name	R/W
31: 00	x	PCIC PIO Error Address	R

9.6 PCI Arbitration and Control

While the microSPARC-IIep CPU is operating as the host and host bridge, the arbitration function for the PCI bus is contained in the PCIC. When the microSPARC-IIep CPU is operating as a non-host bridge subsystem the arbiter is external. As a host bridge, four external masters, in addition to the PCIC, can request the use of the PCI bus. To enable or disable the internal arbiter, refer to Section 9.6.5, *PCIC Arbitration Control Register*

Locking the bus for continuous operations using the LOCK signal is not supported.

As a host arbiter the PCI bus is “parked” on the last master that has been granted the PCI bus.

9.6.1 PCIC Arbitration Assignment Select Register

The PCIC arbitration assignment select register (see *Table 9-38*) is used to select the assignment of request-grant pairs within the PCIC arbiter. The request-grant assignment is used to determine priorities for bus arbitration and allows the priorities to be programmable. However, each assignment must be programmed with a unique value.

Table 9-38 PCIC Arbitration Assignment Select Register (2 bytes @ offset = 8A)

Bit(s)	Reset	Field Name	R/W
14:12 Host agent assignment	100	100: host (internal) assigned as host at level 0 011: req_[3] assigned as host at level 0 010: req_[2] assigned as host at level 0 001: req_[1] assigned as host at level 0 000: req_[0] assigned as host at level 0	R/W
11:09 Agent 3 assignment	011	100: host assigned as Agent 3 at level 2 011: req_[3] assigned as Agent 3 at level 2 010: req_[2] assigned as Agent 3 at level 2 001: req_[1] assigned as Agent 3 at level 2 000: req_[0] assigned as Agent 3 at level 2	R/W
08:06 Agent 2 assignment	010	100: host assigned as Agent 2 at level 2 011: req_[3] assigned as Agent 2 at level 2 010: req_[2] assigned as Agent 2 at level 2 001: req_[1] assigned as Agent 2 at level 2 000: req_[0] assigned as Agent 2 at level 2	R/W
05:03 Agent 1 assignment	001	100: host assigned as Agent 1 at level 1 011: req_[3] assigned as Agent 1 at level 1 010: req_[2] assigned as Agent 1 at level 1 001: req_[1] assigned as Agent 1 at level 1 000: req_[0] assigned as Agent 1 at level 1	R/W
02:00 Agent 1 assignment	000	100: host assigned as Agent 0 at level 1 011: req_[3] assigned as Agent 0 at level 1 010: req_[2] assigned as Agent 0 at level 1 001: req_[1] assigned as Agent 0 at level 1 000: req_[0] assigned as Agent 0 at level 1	R/W

The three-level arbitration algorithm describes the operation of the request and grants as they are in the default condition, which is in response to reset. Programmable assignment of request and grants allows the arbitration priority of all bus masters to be determined by software. The priorities should only be changed when there is no bus activity. The programmable assignment operates for the default round-robin algorithm, even though all bus masters have equal priority in that algorithm.

Note – The PCIC arbitration assignment select register should never be set such that any two loads or the host are assigned the same level and agent. Each assignment must be unique.

9.6.2 PCI Arbitration Algorithm

There are two arbitration algorithms available in the PCIC. Both implement a fairness algorithm as described in revision 2.1 of the PCI specification, on page 56, *Implementation Note: System Arbitration Algorithm*. The first algorithm deals with all possible PCI masters at the same priority level, and rotates a token to the next requestor. In this way all masters are assured of equal access to the PCI bus.

The second algorithm has three levels of assignment for the bus requests (see *Figure 9-6*). Level 0 is the highest priority, with the host processor as the only agent at that level (Agent H). The processor is allocated the bus every other bus operation cycle. Level 1 requests have three agents, representing PCI request 0 (Agent 0), PCI request 1 (Agent 1) and all level 2 requests. When a level 1 agent is granted, and uses, the bus, a token is set representing which level 1 agent last used the bus. All level 1 agents have the same priority, and are granted the bus equally (rotating within level 1). There are two more agents at level 2. The agents at level 2 represent PCI request 2 (Agent 2) and PCI request 3 (Agent 3).

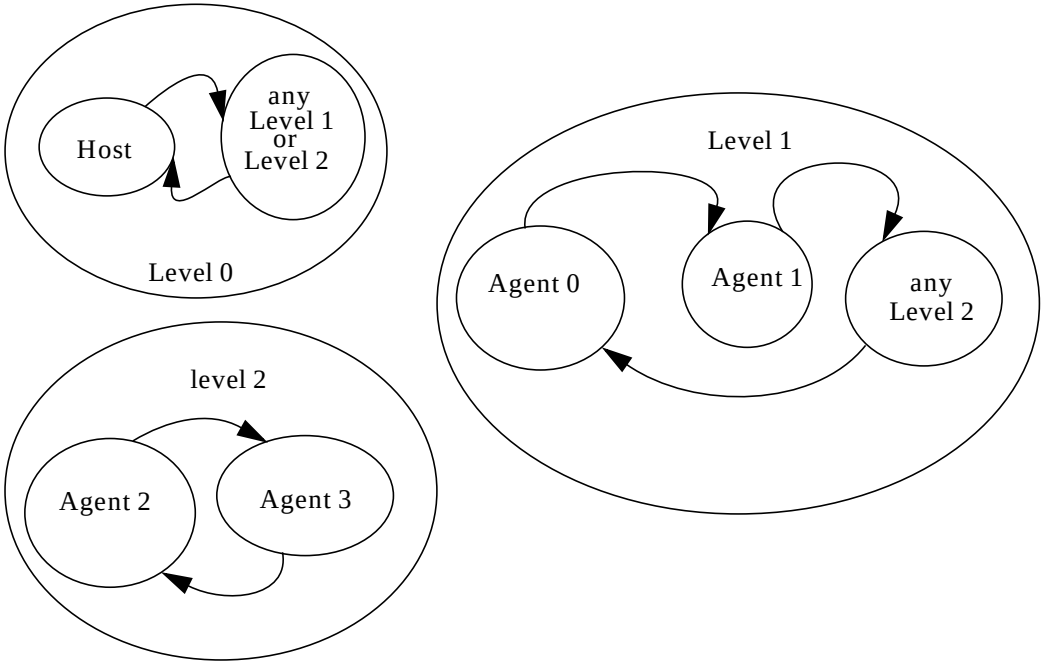


Figure 9-6 Three Level Arbitration Algorithm

9.6.3 PCIC PIO (IAFX Slave) Control Register

The PCIC PIO (IAFX Slave) control register (see *Table 9-39*) is used to control the operation of the PIO (IAFX Slave) interface. This interface accepts transactions from the microSPARC-IIep CPU, buffers the requests in various FIFOs, and dispatches these requests to the PCI bus. Three control bits are defined in *Table 9-39*.

Table 9-39 PCIC (IAFX Slave) PIO Control Register (1 byte @ offset = 60)

Bit(s)	Reset	Field Name	R/W
07	0	PIO Prefetch Enable	R/W
06	0	PIO Burst Enable	R/W
05: 03	0	PIO Reserved	R
02	0	PIO Big-Endian	R/W
01:00	0	PIO Reserved	R

Bit 07 is the Prefetch Enable bit. When enabled, the PCI interface prefetches memory references from the PCI bus. This increases the performance of PIO loads.

Bit 06 is the Burst Enable bit. When this control bit is set, it allows requests for consecutive memory data operations to be packed into a burst on the PCI bus. When set, PIO performance is increased.

Note – It may be required to turn off burst enable when interfacing to very-slow-responding external PCI slave devices, owing to the host's being assigned a low arbitration priority, or for other reasons. This action makes no noticeable difference to the speed of the transfer, since it is already slow. Without burst enable, the CPU to PCIC transfer handshakes on each transaction, rather than implementing transfers in bursts. This mode may prevent timeouts on the IAFX bus. Any reduction in PIO performance is insignificant because that performance was initially poor.

Bit 02 is used to enable big-endian mode on read and write accesses to PIO (IAFX slave) data. When this bit is set, the data and byte enables are not switched to little-endian mode when performing any operations on the PCIC (IAFX Slave) interface. This condition applies to configuration-register reads and writes and PIO. This property may be useful when software operates on big-endian data and requires to maintain big-endian representation when accessing PCIC configuration registers, for example, counters/timers, and the IOTLB. When changing this bit, a store byte should be used to avoid uncertainty as to the current bit setting. In addition, to ensure that all preceding operations have completed prior to changing this bit, a read from this register should be done, discarding the data. Refer to Section 1.3, *microSPARC-IIep Endian Support* on page 4 for more information on endian support and operation.

9.6.4 PCIC DVMA (IAFX Master) Control Register

The PCIC DVMA (IAFX Master) control register (see *Table 9-40*) is used to control the operation of the DVMA (IAFX Master) interface that accepts transactions from the PCI bus, buffers the requests in various FIFOs, and dispatches the requests to the IAFX bus. Three control bits are defined.

Table 9-40 PCIC DVMA (IAFX Master) Control Register (1 byte @ offset = 62)

Bit(s)	Reset	Field Name	R/W
07:05	00	reserved	R
04	0	PCIC DVMA (IAFX Master) Quiescence Acknowledge	R
03	0	reserved	R/W
02	0	reserved	R/W
01	0	PCIC DVMA (IAFX Master) IOTLB Enable	R/W
00	0	PCIC DVMA (IAFX Master) Quiescence request	R/W

Bit 00 and bit 04 are used when the slave PCI interface must be quiescent to allow for memory and I/O activity. When bit 00 is set to one, a request for quiescence is made. After some time, all PCI slave input activity completes (FIFOs are emptied) and any new memory or I/O requests are rejected (retry on PCI). This ensures that any pending memory store operations in the PCI slave input FIFOs are completed. When the quiescent state is reached, it is signaled by the setting of bit 04 of this register.

Note – Quiescence of the PCI bus is needed when entries to the IOTLB are changed. Interrupts may be disabled while requesting quiescence to prevent any additional period of suspension of PCI bus activity to the host.

Bit 01 is used to enable the IOTLB. When set, all addresses that have been accepted by the PCI slave to be used to access main DRAM memory first pass through the IOTLB for translation. When cleared, the IOTLB is bypassed, and addresses are untranslated.

Bits 02 and 03 are reserved and should be set to zero.

9.6.5 PCIC Arbitration Control Register

The PCIC arbitration control register (see *Table 9-41*) is used to control the operation of the internal PCI arbiter and the interrupt controller.

Table 9-41 PCIC Arbitration / Interrupt Control Register (1 byte @ offset = 63)

Bit(s)	Reset	Field Name	R/W
07:05	0	reserved	R
04	0	PCI External Interrupt Controller Select	R / W
03	0	Reserved (must be set to zero)	R / W
02	0 / 1[1]	Internal Arbiter Disable	R / W
01	0	reserved	R
00	0	PCIC Arbitration Level Select	R / W

1. Bit 2 is set to 1 at reset, if the microSPARC-IIep CPU is configured to power up in slave mode (refer to section 9.9). Bit 2 is set to 0 at reset if the microSPARC-IIep CPU is set to power up in host mode.

When bit 00 is cleared, it selects the single level of priority. When it is set it selects the three levels of arbitration. When three-level arbitration is selected, the Arbitration Assignment Select register can be used to map which request / grant pairs are assigned to each level (and must remain uniquely set for each device). Refer to Section 9.6.1, *PCIC Arbitration Assignment Select Register*.

Bit 02 is used to disable the internal arbiter. When this is done, an external arbiter is required to resolve bus requests. When the internal arbiter is disabled:

- The microSPARC-IIep CPU signals a request to use the bus on the output pin PCI_GNT_L[0] and receives an acknowledgment from the external arbiter on the input pin PCI_REQ_L[0]. (The direction of these signals is independent of the internal arbiter enable state. However, the interpretation of these signals changes with the state of the internal arbiter disable bit.) Bit 2 is set to 1 when reset occurs, if the microSPARC-IIep CPU has pins PLL_BYP_L and EXT_CLK2 set to power up in slave mode (refer to Section 9.4.2, *PCI Host/Satellite Mode* and Section 9.9, *System Status and System Control*). Bit 2 is set to 0 when reset occurs if the microSPARC-IIep CPU has these pins set to power up in host mode.
- The PCI_GNT_L[1] output pin signals when the microSPARC-IIep CPU is requesting to use the PCI bus for an extended operation. An extended operation is requested for PCI configuration cycles (IDSEL bus charging), when quiescence has been requested (preventing retries from occurring too often), or when the PCI host is requesting the bus to be parked. During an extended operation request, the bus activity may not start immediately following the bus grant signal.
- The PCI_GNT_L[2] output pin signals when the microSPARC-IIep CPU has detected any unmasked internally detected interrupts. When the internal interrupt controller is bypassed (refer to bit 04) and an external interrupt controller is used to drive the IRL lines directly into the microSPARC-IIep CPU, this output pin could be monitored by the external controller. This signal is asserted when any unmasked lever I5 interrupt is signaled or when an unmasked timer interrupt is signaled.

Bit 03 is reserved and must be set to zero.

Bit 04 selects an external interrupt controller and bypasses the internal interrupt controller. In this case the four PCI interrupt signals (INTD/C/B/A or PCI_INT_L[3:0]) are routed directly to the four internal interrupt request lines (IRL) of the microSPARC-IIep CPU. These IRL lines are those that interface directly to the standard SPARC version 8 interrupt request inputs:

INTD#->IRL[3]

INTC#->IRL[2]

INTB#->IRL[1]

INTA#->IRL[0]

The CPU samples the IRL lines at each cycle and, if they are seen unchanged for two cycles, responds to the interrupt request following the standard SPARC interrupt priority. When the internal interrupt controller is bypassed, an external interrupt controller could be used to provide up to 15 levels of interrupt to the microSPARC-IIep CPU. Refer to the description of bit 02 for how to access the interrupts that are detected internally when an external interrupt controller is used. For more information, refer to Section 9.7.2, *PCIC System Interrupt Pending Register*.

9.7 PCIC Interrupts

The PCIC also contains interrupt control logic that receives interrupts from the PCI bus (INTD#/C#/B#/A# or PCI_INT_L[3:0]) and generates an interrupt vector to the microSPARC-IIep core. The same interrupt pins may be used as outputs of signal interrupt conditions to external devices if the internal interrupt controller is disabled. In addition, any internally detected error conditions generate a level 15 interrupt vector. Interrupt vectors in the microSPARC-IIep processor are processed according to the normal SPARC interrupt structure. There are several configuration registers in the PCIC devoted to interrupt control. These registers provide the same function as the interrupt control unit of the 89C105 interrupt controller.

The PCI interrupts are *level* signals, and must be held low for a minimum of two processor clocks before the IRL lines appear stable to the processor to ensure that it responds to the interrupt.

9.7.1 PCIC Interrupt Assignment Select Registers

The PCIC interrupt assignment select registers—see *Table 9-42* and *Table 9-43*—are used to assign interrupt input signals a chosen priority level. When this is done, subsequent masking operations apply to the assigned interrupt priority level. Each interrupt can be mapped to any priority level, independently of other interrupt assignments. More than one interrupt can be assigned the same priority level, which requires that the software interrupt handler determine which interrupt occurred.

Table 9-42 PCIC Interrupt Assignment Select Register (2 bytes @ offset = 88)

Bit(s)	Reset	Field Name	R/W
15:12	0x7	PCI INTD# (PCI_INT_L[3]) assignment field	R/W
11:08	0x5	PCI INTC# (PCI_INT_L[2]) assignment field	R/W
07:04	0x3	PCI INTB# (PCI_INT_L[1]) assignment field	R/W
03:00	0x2	PCI INTA# (PCI_INT_L[0]) assignment field	R/W

Table 9-43 PCIC Interrupt Assignment Select Register (2 bytes @ offset = 8C)

Bit(s)	Reset	Field Name	R/W
15:12	0x0	PCI_INT_L[7] assignment field	R/W
11:08	0x0	PCI_INT_L[6] assignment field	R/W
07:04	0x0	PCI_INT_L[5] assignment field	R/W
03:00	0x0	PCI_INT_L[4] assignment field	R/W

Any interrupt assigned priority zero is disabled, since an IRL code of zero signals the absence of any interrupt to the processor.

Figure 9-7 shows a block diagram of the PCIC interrupt controller.

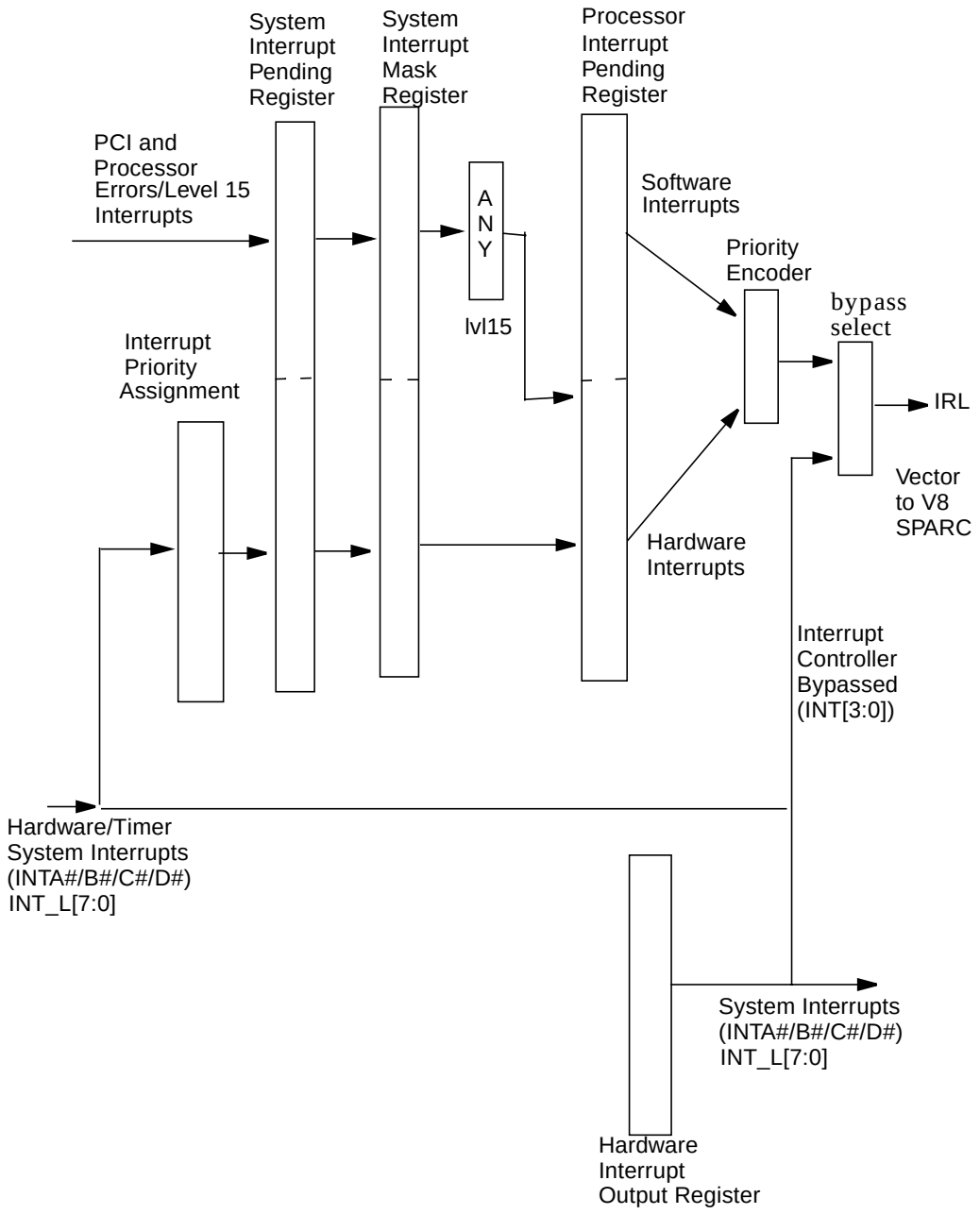


Figure 9-7 PCIC Interrupt Controller Block Diagram

9.7.2 PCIC System Interrupt Pending Register

The PCIC System Interrupt Pending register, described in *Table 9-44*, is used to read status information for any pending system—that is, hardware associated—interrupts. The state of the eight PCI interrupt lines and the two timer interrupts can be examined by reading this register. Any error conditions detected by the PCIC that result in a level 15 interrupt are also signaled by posting bits in this register.

Table 9-44 PCIC System Interrupt Pending Register (4 bytes @ offset = 70)

Bit(s)	Reset	Field Name	R/W
31	0	reserved; read as zero	R
30	0	PCIC PIO Detected Error	R
29	0	PCIC DMA Detected Error	R
28	0	PCI Bus Error (SERR#) signaled	R
27	0	Processor Detected Error (AFSR or MFSR)	R
26	0	PCI Reset detected	R
25:16	0	Reserved. Read as zero	R
15:1	0	Assigned HW Interrupts	R
0	0	reserved; read as zero	R

Bits 1–15 reflect the state of the programmable, priority-assigned hardware interrupts which are the eight assigned PCI interrupts, and the two assigned timer interrupts. The PCI interrupts are level sensitive, active low, and are defined by the PCI specification to remain active until some processor action clears them. The interrupt controller does not resynchronize these signals but it does perform the assignment, masking and comparison to the software interrupt level before passing the signals through to the processor IRL lines. The processor IRL lines are sampled for two clocks to avoid reacting to glitches.

When the bypass path is selected, the four PCI interrupt pins are routed directly to the processor IRL lines:

PCI_INT[3] → IRL[3]
PCI_INT[2] → IRL[2]
PCI_INT[1] → IRL[1]
PCI_INT[0] → INT[0]

where the processor samples them for two processor clocks (not PCI clocks) to ensure that they are stable before responding. When the interrupt controller is bypassed, the internal interrupt condition may not be available to the external interrupt controller. Consequently, he internal timer interrupts and any error conditions detected by the PCIC—or by the microSPARC-IIep CPU— may not be

able to generate a signal to the external interrupt controller. Refer to the description of the arbiter-disable bit in Section 9.6.5, *PCIC Arbitration Control Register* for a method of monitoring these conditions externally when the internal arbiter is disabled.

Bit 26 is set whenever the processor detects an active PCI reset input signal while the PCI RESET pin is used as an input. If the PCI input pin is enabled as a processor reset, bit 26 is cleared as a result of that reset—see Section 9.9.1, *System Status and System Control (Reset) Register*. Bit 26 has a latching memory effect and is set on the first detection of the PCI reset signal. While PCI RESET is asserted, bit 26 cannot be cleared by writes to the clear system interrupt pending register. This bit remains set after the PCI RESET input signal is removed until it is cleared with reset or by a write to the Clear System Interrupt Pending register after the reset condition has been removed. See Section 9.8, *Counter-Timers*.

Note – The PCI reset input, when enabled as a processor reset, overrides the level 15 interrupt that is set when bit 26 is set.

Bit 27 is set whenever the processor detects an internal level 15 interrupt condition that results in bit 31 of the Asynchronous Fault Status register (AFSR) or bit 31 of the Memory Fault Status register (MFSR) being set. Refer to Section 5.9, *Control Space MMU Registers* for details on the AFSR and the MFSR.

Bit 28 is set whenever any subsystem on the PCI bus signals SERR#. The interrupt is generated and does not depend on which particular PCI subsystems were involved in a transaction or if any were involved at all. Polling the PCI configuration registers of all devices on the PCI bus may be necessary to determine the cause of the SERR# signal. Signaling of SERR# by the PCIC itself can be disabled—see Section 9.5.2.2, *PCI Device Control*.

Bit 29 is set whenever an error is detected on a PCI DMA operation. This error can occur for an IOTLB miss while the IOTLB is enabled. When bit 29 is set, the PCI virtual address that would have resulted in a IOTLB miss is saved in the IOTLB translation error address register (0xCC).

Bit 30 is set when an error condition is detected on a PIO transaction that is terminated abnormally. Additional status information may be present in the PCI Device Status Configuration register (0x06) and in external PCI device status registers—see Section 9.5.2.3, *PCI Device Status* on page 162. When bit 30 is set, the command and address that applied when the error occurred is saved in the PCI Master Error Command register and the PCI Master Error Address register. See Section 9.5.9, *PCIC PIO Error Command and Address Registers*.

9.7.3 PCIC Clear System Interrupt Pending Register

The PCIC Clear System Interrupt Pending register (*Table 9-45*) is used to clear any system PCIC interrupts set as a result of an error. Only error conditions detected by the PCIC and which resulted in a level 15-nonmaskable interrupt can be cleared by setting bits in this register. These bits cannot be used to clear the state of the four PCI interrupt lines.

Table 9-45 PCIC Clear System Interrupt Pending Register (1 byte @ offset = 83)

Bit(s)	Reset[1]	Field Name	R/W
07	0	Clear all PCIC Detected system interrupt pending Level 15 Errors	W
06	0	Clear PCIC PIO Detected Error	W
05	0	Clear PCIC DMA Detected Error	W
04	0	Clear PCI SERR# signaled	W
03	0	Clear PCI Reset Signaled	W
02: 00	000	reserved; read as zero	W

1. Writing a one to the bit positions in this register clears the bit in the System Interrupt Pending register.

The ASFR and the MFSR interrupts are not cleared using this register. Refer to Section 5.9, *Control Space MMU Registers*, for how to clear these interrupts.

- Bit 07, when set, clears all system interrupt pending interrupts that are set as a result of a PCIC detected error condition. This has the same effect as turning on bits 4 through 6.
- Bit 06, when set, clears the PCIC PIO detected error.
- Bit 05, when set, clears the IOTLB translation error.
- Bit 04, when set, clears the PCI SERR# interrupt.
- Bit 03, when set, clears the PCI Reset interrupt.

9.7.4 PCIC System Interrupt Target Mask Register

The PCIC system interrupt target mask register occupies three addresses

- 0x74, for reading the current state of the interrupt mask; see *Table 9-46*.
- 0x7C, for setting the interrupt mask bits; see *Table 9-47*.
- 0x78, for clearing the mask bits; see *Table 9-48*.

Table 9-46 PCIC System Interrupt Target Mask Register (4 bytes @ offset = 74)

Bit(s)	Reset	Field Name	R/W
31	1	Mask All Interrupts, HW and/or SW	R
30	1	Mask PCI PIO Detected Error	R
29	1	Mask PCI DMA Detected Error	R
28	1	Mask PCI SERR# Signaled	R
27	1	Mask Processor Detected Error (AFSR or MFSR)	R
26	1	MASK PCI Reset (as input) Detected	R
25:16	0	reserved; read as zero	R
15:01	0x7f	Mask Assigned HW Interrupts	R
00	0	reserved; read as zero	R

Table 9-47 PCIC System Interrupt Target Mask Clear Register (4 bytes @ offset = 78)

Bit(s)	Reset	Field Name	R/W
31	0	Clear Mask All Interrupts, HW and/or SW	W
30	0	Clear Mask PCI PIO Detected Error	W
29	0	Clear Mask PCI DMA Detected Error	W
28	0	Clear Mask PCI SERR# Signaled	W
27	0	Clear Mask Processor Detected Error (AFSR or MFSR)	W
26	0	Clear Mask PCI Reset (as input) Detected	W
25:16	0	reserved; read as zero	W
15:01	0x0	Clear Mask Assigned HW Interrupts	W
00	0	reserved; read as zero	W

Table 9-48 PCIC System Interrupt Target Mask Set Register (4 bytes @ offset = 7C)

Bit(s)	Reset	Field Name	R/W
31	0	Set Mask All Interrupts, HW and/or SW	W
30	0	Set Mask PCI PIO Detected Error	W
29	0	Set Mask PCI DMA Detected Error	W
28	0	Set Mask PCI SERR# Signaled	W
27	0	Set Mask Processor Detected Error (AFSR or MFSR)	W
26	0	Set Mask PCI Reset (as input) Detected	W

Table 9-48 PCIC System Interrupt Target Mask Set Register (4 bytes @ offset = 7C)

Bit(s)	Reset	Field Name	R/W
25:16	0	reserved; read as zero	W
15:01	0x0	Set Mask Assigned HW Interrupts	W
00	0	reserved; read as zero	W

Writing a one to any defined bit field in the mask set register disables that interrupt, and writing a one to the same field in the mask clear register re-enables it. All pending interrupts are cleared, and all mask bits are set upon system reset. (The state of the PCI_INT_L[7:0] lines that are driven by external sources are defined by that external source. However the interrupt is masked in the PCIC if the external source were driving it.)

The interrupts in bit positions 15:01 refer to those hardware interrupts that have had their mapping priority assignment performed.

9.7.5 PCIC Processor Interrupt Pending Register

The PCIC Processor Interrupt Pending register (see *Table 9-50*) is used to read status information for any pending processor interrupts. These can be hardware or software interrupts. This status reflects the state of the currently unmasked interrupts. The highest-priority, unmasked, hardware or software interrupt is the one that generates the vector to the microSPARC-IIep processor. (The PCI RESET input detected is latched and held until cleared.)

Note – The highest priority interrupt generates an interrupt vector code to the microSPARC-IIep processor which has the standard SPARC version 8 interrupt processing features to process that interrupt. Interrupt inputs on the processor IRLs are sampled using two clocks to verify that these inputs are stable.

Using the default (reset) assignment interrupt mapping priorities, the interrupts are assigned as shown in *Table 9-49*.

Table 9-49 PCIC Default (Reset) Interrupt Assignments

Interrupt Priority Level	Hardware Interrupt
15[1]	PCI PIO Detected Error
15 [1]	PCI DMA Detected Error
15 [1]	PCI Bus (SERR#) Detected Error
15 [1]	Processor Detected Error (AFSR,MFSR)

Table 9-49 PCIC Default (Reset) Interrupt Assignments *(Continued)*

Interrupt Priority Level	Hardware Interrupt
15 [1]	PCI reset (as input) detected
7	PCI_INTD# (PCI_INT_L[3])
5	PCI_INTC# (PCI_INT_L[2])
3	PCI_INTB# (PCI_INT_L[1])
2	PC_INTA# (PCI_INT_L[0])
0 (Disabled)	PCI_INT_L[7]
0 (Disabled)	PCI_INT_L[6]
0 (Disabled)	PCI_INT_L[5]
0 (Disabled)	PCI_INT_L[4]
0 (Disabled)	Processor Counter Interrupt
0 (Disabled)	System Counter Interrupt

1. not reassignable

Table 9-50 PCIC Processor Interrupt Pending Register (4 bytes @ offset = 64)

Bit(s)	Reset	Field Name	R/W
31:17	0	Software Interrupts	R
16	0	reserved; read as zero	R
15:01	0	Assigned Unmasked HW Interrupts	R
00	0	reserved; read as zero	R

9.7.6 PCIC Software Interrupts

There are two registers that allow software to generate and clear software interrupts. The software interrupts are read from the processor interrupt pending register. However, the software interrupts are cleared by writing a one to the appropriate bit positions in the PCIC Software Interrupt Clear register (0x6A, see *Table 9-51*) or set by writing to the PCIC software interrupt set register (0x6E, see *Table 9-52*).

Table 9-51 PCIC Software Interrupt Clear Register (2 bytes @ offset = 6A)

Bit(s)	Reset	Field Name	R/W
15:01	0	Clear Software Interrupt	W
00	0	reserved; read as zero	W

Table 9-52 PCIC Software Interrupt Set Register (2 bytes @ offset = 6E)

Bit(s)	Reset	Field Name	R/W
15:01	0	Set Software Interrupt	W
00	0	reserved; read as zero	W

9.7.7 PCIC Hardware Interrupt Outputs

The microSPARC-IIep CPU can signal interrupts to an external controller or it can drive output status lines directly under processor register control. Each of the eight interrupt lines can be driven as an output. These open-drain drivers can be driven by multiple sources and require an external pull-up.

Note – Interrupt lines that are not used to signal interrupts to the microSPARC-IIep CPU can be used as output interrupts to another processor or they can be used to control other functions under direct microSPARC-IIep program control. Such programmable functions include system status indicators, control selectors, and inter-processor interrupts.

The interrupt output lines are of open-drain configuration, and allow for multiple sources to drive the signal. Even when the microSPARC-IIep CPU has cleared a bit of the hardware interrupt output register, other external devices may keep the signal active.

The PCIC hardware interrupt output register (see *Table 9-53*) generates hardware interrupt outputs. When a bit in this register is set, the interrupt output signal of the microSPARC-IIep CPU is activated. When a bit in this register is in a cleared state, the microSPARC-IIep CPU does not activate the external pin. In addition, the signaling of an interrupt on the external pin activates the microSPARC-IIep CPU's input interrupt detection circuit. If the mask bit of the input interrupt is set, however, that interrupt does not result in the microSPARC-IIep CPU's taking the interrupt that it generated.

Table 9-53 PCIC Software Interrupt Output Register (1 byte @ offset = 8E)

Bit(s)	Reset	Field Name	R/W
07	0	PCI_INT_L[7] Enable Interrupt	R/W
06	0	PCI_INT_L[6] Enable Interrupt	R/W
05	0	PCI_INT_L[5] Enable Interrupt	R/W
04	0	PCI_INT_L[4] Enable Interrupt	R/W
03	0	PCI INTD (PCI_INT_L[3]) Enable Interrupt	R/W

Table 9-53 PCIC Software Interrupt Output Register (1 byte @ offset = 8E) *(Continued)*

Bit(s)	Reset	Field Name	R/W
02	0	PCI INTC (PCI_INT_L[2]) Enable Interrupt	R/W
01	0	PCI INTB (PCI_INT_L[1]) Enable Interrupt	R/W
00	0	PCI INTA (PCI_INT_L[0]) Enable Interrupt	R/W

9.8 Counter-Timers

The microSPARC-IIep CPU features two programmable counter-timers designed to provide a system timer and a single processor-specific timing function. The features of these two counter-timers are similar to those offered in the SLAVIO chip (STP2001 Slave I/O Controller) used with the microSPARC-II. The 31-bit system counter is dedicated to the system timer function, and it generates an interrupt upon time-out. The processor counter can either be configured to behave as a 31-bit timer that generates an interrupt upon time-out or it can provide a real-time 63-bit counter for high-resolution user-performance analysis.

When the processor counter is set up to behave as a 31-bit timer, it can be used for OS kernel profiling. In the 63-bit-counter mode, the timer can be loaded upon each entry into user mode and saved on exit. It can also be loaded with a binary real-time value to track time-of-day precisely.

These registers should only be accessed in word mode. There are restrictions placed on the access sequence for the user timer. That register requires two word accesses performed as one snapshot operation by the hardware to prevent the software-visible counter from ticking between these accesses.

These timers tick once every four processor clocks. A block diagram of the system counter and processor counter/user timer is presented in *Figure 9-8*.

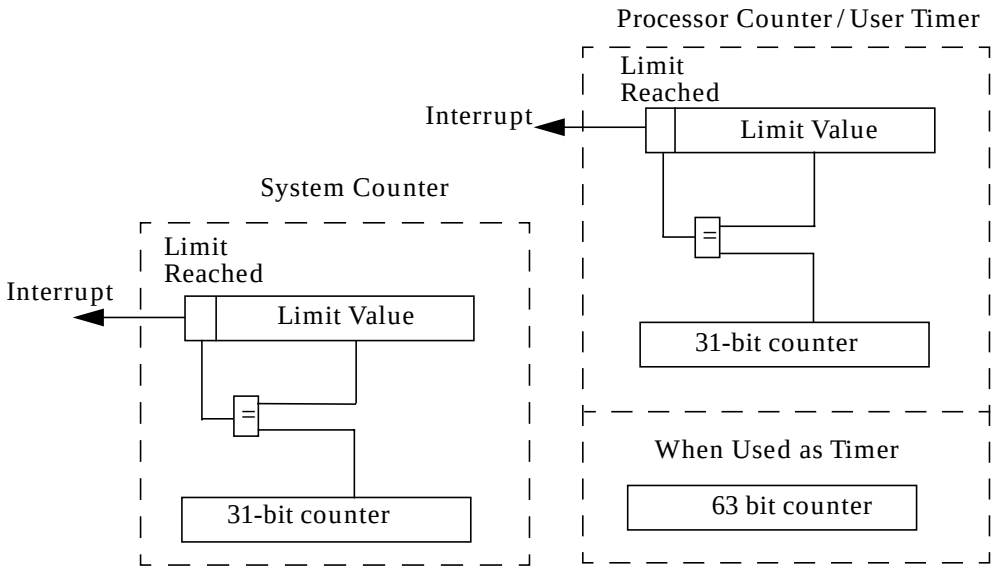


Figure 9-8 Counter-Timer Block Diagram

9.8.1 Counter-Timers Address Map and Function

Three addresses are associated with each counter: a count register, a limit register, and a pseudo register that allows the limit to be loaded without resetting the count. The registers are described in the following sections.

Each counter increments by one in bit in position 0 every four processor clocks. When the counter reaches the value in its corresponding limit register, it is reset to 0x1, the *limit-reached* bits in both the counter and the limit registers are set, and an interrupt is generated (if enabled) at the interrupt level specified in the Counter Interrupt Priority Assignment register.

The interrupt is cleared and the limit bits reset by reading the appropriate limit register. Reading the counter register does not change the state of the limit bit. Writing the limit register resets the corresponding counter to 0x01.

The limit register can be loaded via the pseudo register without resetting the count.

If the count value is already higher than the new limit, the counter counts to its maximum value, then resets and counts up to the new limit value before generating the interrupt. This property allows alarm-clock, rather than time-tick, usage of the counter.

Setting the limit register to 0x0 causes the corresponding counter to free-run. In this case, an interrupt is generated when the counter overflows. All bits in the limit register are cleared to zero on reset, and the counter is set to the value 0x01.

Table 9-54 shows the address map of the PCIC counter-timers.

Table 9-54 PCIC Counter-Timers Address Map

Address Offset	Size	Register	R/W
0xAC	word	Processor Counter Limit register or User Timer MSW	R/W
0xB0	word	Processor Counter register or User Timer LSW	R/W
0xB4	word	Processor Counter Limit register (non-resetting port)	W
0xB8	word	System Limit register	R/W
0xBC	word	System Counter register	R/W
0xC0	word	System Limit register (non-resetting port)	W
0xC4	byte	Processor Counter User Timer Start/Stop Register	R/W
0xC5	byte	Timer Configuration Register	R/W
0xC6	byte	Counter Interrupt Priority Assignment Level register	R/W

9.8.2 Processor Counter Limit Register or User Timer MSW

The Processor Counter Limit register or User Timer most significant word (see *Table 9-55*) occupy the same address decode.

Table 9-55 Processor Counter Limit or User Timer MSW (Word only @ offset = AC)

Processor Counter Mode	Bit(s)	Reset	Field Name	R/W
Counter Mode	31	0	Processor Counter Limit Reached	R
Counter Mode	30:00	0	Processor Counter Limit register	R/W
Timer Mode	31:00	0	User Timer Most Significant Word (MSW)	R/W

The processor counter *limit-reached* bit is set when the processor counter matches the Processor Counter Limit register. It is cleared by reading the Processor Counter Limit register.

The User Timer most significant word (MSW) contains a snapshot of the user timer register. The User Timer register is a 64-bit value and can only be read by reading two 32-bit registers. The user timer MSW contains a snapshot of the timer value when the user timer least significant word (LSW) was read. This allows a full 64-bit value to be reflected in the two 32-bit reads. The user timer LSW should always be read first, which operation also transfers the value from the timer MSW into the user timer MSW. Reading from the user timer MSW releases the snapshot and allows the shadow register to be reloaded.

Writing to the user timer also involves transferring from two 32-bit registers into the user timer. There is a sequence required for this write to occur as one update to the user timer (see *Table 9-56*). The user timer MSW should be loaded first, which causes a holding register to load. When the user timer LSW is written, the contents of the MSW holding register, along with the LSW are written into the full 64-bit user timer register.

Table 9-56 User Timer Read / Write Sequence Required

User Timer 64 bit operation	Read	Write
User Timer Most Significant Word (MSW)	2nd	1st
User Timer Least Significant Word (LSW)	1st	2nd

9.8.3 Processor Counter Register or User Timer LSW

The processor counter register or user timer least significant word (see *Table 9-57*) occupy the same address decode.

Table 9-57 Processor Counter or User Timer LSW (Word Only @ offset = B0)

Processor Counter Mode	Bit(s)	Reset	Field Name	R/W
Counter Mode	31	0	Processor Counter Limit Reached	R
Counter Mode	30:00	0	Processor Counter Register	R / W
Timer Mode	31:00	0	User Timer Least Significant Word (LSW)	R / W

The Processor Counter Limit-reached bit is set when the processor counter matches the Processor Counter Limit register. It is cleared by reading the Processor Counter Limit register.

The User Timer least significant word (LSW) triggers a snapshot of the User Timer register. The User Timer register holds a 64 bit value that can only be read by reading two 32-bit registers. The User Timer MSW contains a snapshot of the timer taken at the time that the User Timer least significant word (LSW) was read. This allows a full 64 bit value to be reflected in the two 32-bit reads. The User Timer LSW should always be read first, which operation also transfers the value from the timer MSW into the User Timer MSW. Reading from the User Timer MSW releases the snapshot and allows the shadow register to be reloaded.

Writing to the User Timer also involves transferring from two 32-bit registers into the User Timer. There is a sequence required for this write to happen as one update to the User Timer. The User Timer MSW should be loaded first, which actually loads a holding register. When the User Timer LSW is written, the contents of the MSW holding register, along with the LSW are written into the full 64-bit User Timer register

Refer to Section 9.8.2, *Processor Counter Limit Register or User Timer MSW* for the sequence required to read or write the 64-bit user timer.

9.8.4 Processor Counter Limit Pseudo Register

The Processor Counter Limit Pseudo register (see *Table 9-58*) allows the limit register to be reloaded without resetting the counter. This is a write-only register location. Reads from this register return zeros.

Table 9-58 Processor Counter Limit Pseudo Register (Word Only @ offset = B4)

Bit(s)	Reset	Field Name	R/W
31:00	0	Processor Counter Limit Pseudo register	W

9.8.5 System Counter Limit Register

The System Counter Limit register (see *Table 9-59*) operates similarly to the Processor Counter Limit register. The system counter *limit-reached* bit is set when the system counter matches the system counter limit register. It is cleared by reading the System Counter Limit register.

Table 9-59 System Counter Limit Register (Word Only @ offset = B8)

Bit(s)	Reset	Field Name	R/W
31	0	System Counter Limit Reached	R
30:00	0	System Counter Limit register	R/W

9.8.6 System Counter Register

The system counter register (see *Table 9-60*) operates similarly to the processor counter register.

Table 9-60 System Counter Register (Word Only @ offset = BC)

Bit(s)	Reset	Field Name	R/W
31	0	System Counter Limit Reached	R
30:00	0	System Counter register	R/W

The system counter limit-reached bit is set when the system counter matches the system counter limit register. It is cleared by reading the system counter limit register.

9.8.7 System Counter Limit Pseudo Register

The system counter limit pseudo register (see *Table 9-61*) allows the limit register to be reloaded without resetting the counter. This register is write-only; an attempt to read it returns zeros.

Table 9-61 System Counter Limit or User Timer MSW (Word Only @ offset = C0)

Bit(s)	Reset	Field Name	R/W
31:00	0	System Counter Limit Pseudo register	W

9.8.8 User Timer Start/Stop Register

The user timer start/stop register (see *Table 9-62*) controls the user timer operation, and therefore only operates in user-timer mode. When bit zero is set, the user timer is enabled; when reset to zero, the timer is frozen.

Table 9-62 User Timer Start/Stop Register (1 byte @ offset = C4)

Bit(s)	Reset	Field Name	R/W
07:01	0	unused; read as zero	R
00	0	User Timer Run Enable	R/W

The user timer start/stop register is provided to allow fast trap handlers to stop the user timer blindly while processing time-critical code, without having to read and save the count value. The timer must be restarted before reentering user state. A software flag must be maintained to indicate whether the user timer is in use, so that the fast trap handler may see that it must be restarted. This register has no effect if the processor counter is configured as a counter.

9.8.9 Processor Counter or User Timer Configuration Register

The processor counter or user timer configuration register (see *Table 9-63*) controls the mode of operation of the processor counter and allows writes to the counter registers to affect the counter value.

Table 9-63 Processor Counter/User Timer Configuration Register (1 byte @ offset = C5)

Bit(s)	Reset	Field Name	R/W
07	0	Allow Writes to System Counter	R/W
06	0	Allow Writes to Processor Counter	R/W
05:01	0	unused; read as zero	R
00	0	User Timer Mode Enable	R/W

When bit 0 is set, the processor counter operates in the user-timer mode. When bit 0 is reset, the processor counter operates in counter mode.

When bit 6 is set, write operations to the processor counter result in the counter's being written. This allows diagnostic testing of the counter operation. When bit 6 is reset, writes to the counter are disabled and have no effect on the counter.

Note – Bit 6 is not used in the user timer mode and writes cannot be disabled in this mode.

When bit 7 is set, write operations to the system counter result in the counter's being written. This allows diagnostic testing of the counter operation. When bit 7 is reset, writes to the counter are disabled and have no effect on the counter.

9.8.10 Counter Interrupt Priority Assignment Register

The Counter Interrupt Priority Assignment register (see *Table 9-64*) controls the priority level for the Processor-counter and the System-counter interrupts.

Table 9-64 Counter Interrupt Priority Assignment Register (1 byte @ offset = C6)

Bit(s)	Reset	Field Name	R/W
07:04	0	System Counter Interrupt Priority Level	R/W
03:00	0	Processor Counter Interrupt Priority Level	R/W

When the processor counter operates in user timer mode, it cannot generate an interrupt.

Bits 07:04 assign the system counter interrupt priority. This interrupt priority is used for masking and generation of hardware interrupt levels for the system counter. This register works similarly to the interrupt assignment register for PCI interrupts. Multiple interrupts can be assigned the same priority level, but then require software to determine the source of the interrupt. Assigning a interrupt priority level of 0 disables the interrupt.

Bits 03:00 assign the processor counter interrupt priority. The interrupt priority assigned is used for masking and generation of hardware interrupt levels for the processor counter. Assigning a interrupt priority level of 0 disables the interrupt. When the processor counter is in the user timer mode, it cannot generate an interrupt.

9.9 System Status and System Control

The system status and system control section of the PCIC is used to control initialization of the microSPARC-IIep CPU. The source of a processor reset leaves bits set in the System Status and System Control (Reset) register. Other bits in that register are used to direct the processors operation for reset operations.

9.9.1 System Status and System Control (Reset) Register

The System Status and System Control (Reset) register is used to report what kind of reset the processor last experienced. Since reset places the processor in a known state, most control bits in all registers become defined in response to a reset. The Reset register allows the processor to read which reset type was last recorded. In addition this register allows the processor to set a bit that simulates a system reset (software reset) and select a response to a received PCI reset

Table 9-65 System Status and System Control Register: 1 byte @ offset = D0

Bit(s)	Reset	Field name	R/W
07	0/1[1]	PCI Slave Mode Pin Setting (slave mode = 1)	R
06	0/1 [1]	Enable PCI Input Reset (slave mode only)	R/W
05	0/1	PCI Input Reset Status	R/C
04	0	Processor Watchdog Reset	R/C
03	0/1	Input Reset Status (power-up)	R/C
02	0	reserved	R
01	0	Software Reset Status	R/C
00	0	Software Reset Control	W

1. Reflects the hardwired strapping of the mode control pins—see text

Bit 07 is the PCI Slave Mode Pin Setting bit. It reflects the hardwired strapping of the mode control pins that place the microSPARC-IIep CPU in a PCI host master mode (bit =0), or a PCI slave mode (bit = 1). In the host mode, the microSPARC-IIep CPU drives the PCI reset pin as an output signal. In the slave mode, the microSPARC-IIep CPU can respond to that pin as an input.

Note – The microSPARC-IIep CPU can only be placed in slave mode when the phase-locked loop is selected as the clock source. The PLL_BYP_L pin, when tied high, selects the normal PLL mode of operation, and allows the EXT_CLK2 pin to select slave mode (EXT_CLK2 tied high) or master mode (EXT_CLK2 tied low). When the PLL_BYP_L pin is tied low, the PLL is bypassed and EXT_CLK2 is used to generate CPU clocks. In this case the PCI Slave Mode Pin Setting bit displays master mode as the default when reset.

Bit 06 is an enable to allow the PCI reset input pin to cause a processor reset. When bit 06 is set to a one, and the microSPARC-IIep CPU is in slave mode (bit 07 = 1), a PCI reset forces a reset of the microSPARC-IIep CPU, and sets the PCI Reset Status bit (bit 05). When Bit 06 is set to a zero, or if the microSPARC-IIep CPU is not in slave mode (the microSPARC-IIep CPU is the source of the PCI reset signal), PCI reset has no reset effect on the processor's internal state. (Note that while the PCI bus is being reset, the processor does not receive responses from accesses to it. Refer to the interrupt section for a description of the level-15 interrupt effects due to a PCI reset.) Note that power-on reset sets bit 06 to match the value of bit 07. Watchdog reset has no effect on this bit.

Bit 05 is the PCI reset indicator, and is set in response to an input PCI reset if enabled by bit 06. This read-only bit is set when a PCI reset is received while a slave and PCI reset has been enabled (bit 06 = 1). This bit is cleared by a power-on reset or by writing a '0' to it. Writing a '1' to it has no effect.

Note – Bit 05 is set following power up reset if the PCI reset input remains active after the input_reset_l signal is removed. Bit 05 is reset if the PCI reset input is removed before the input_reset_l signal is removed.

Bit 04 is the Watchdog Reset indicator. This bit is set to a 1 when a watchdog reset is initiated. See section 10 for details on what initiates a watchdog reset. (Inside the processor, a watchdog reset is signaled with the IU_ERROR signal.) This bit is cleared by either a power-on reset, a PCI reset, a software reset, or by writing a zero to this bit. Writing a one has no effect. Note: a watchdog reset does not propagate out to the PCI bus but remains within the microSPARC-IIep CPU.

Bit 03 is the Input Reset (Power-up) indicator. This bit is set to a one when the input_reset_l signal is activated. This bit is cleared by writing a zero to this bit. Writing a 1 has no effect.

Bit 02 is reserved, and should not be written.

Bit 01 is the Software Reset indicator. This bit is set to 1 when a software reset has been initiated (setting bit 00 = 1). Software Reset has the same effects on the processor state as power-on reset, with the exception that a software reset sets this bit. This bit is cleared by a power-on reset, a PCI reset, or by writing a 0 to it. Writing a 1 has no effect.

Bit 00 is the Software Reset (write-only) bit. When set to a ‘1’, this generates the equivalent of a power-on reset. When the microSPARC-IIep CPU operates as the host processor, a power-on reset drives the reset output to the PCI bus. When the microSPARC-IIep CPU operates in slave mode, it does not drive the PCI reset pin, but may accept this bit as an input signal.

9.10 PCI Interface Signal Description

The PCI signal definitions are listed alphabetically. The PCIC interface does not implement IDSEL, as it does not support PCI configuration cycle access from the PCI bus. The LOCK signal is also not used, and there is no support for PCI locked operations.

Table 9-66 PCI Signal Listing

SIGNAL NAME	PINS	ACTIVE	I/O	DESCRIPTION
AD[31:0] Address / Data	32	HIGH	O I	AD[31:0] drive the physical address during the first clock of FRAME# during a transaction. During subsequent cycles, AD[31:0] contain write data. AD[31:24] define the most significant byte, and AD[7:0] define the least significant byte. It represents the physical address to be decoded as a check for being the target of the current transaction, and also for receiving data.
C / BE#[3:0] Command / Byte Enables	4	HIGH / LOW	O I	During the address phase, C / BE[3:0] define the bus command (see section 5.2), which is asserted high. During the data phase C / BE[3:0] are byte enables, which are asserted LOW. BE[3] applies to the most significant byte (AD[31:24]). It indicates the command another PCI master is executing, or the byte enables for input data.
DEVSEL# Device Select	1	LOW	O I	When asserted, it indicates that the PCIC has decoded the current physical address and is the target of the current transaction. When negated, the PCIC has decoded that it is not the target of the current transaction. When asserted it indicates some other PCI device has been selected. When negated, it indicates that no other device has been selected.
FRAME# Frame	1	LOW	O	When first asserted FRAME# indicates that a bus transaction is beginning. While FRAME# remains asserted, data transfers continue. When negated FRAME# indicates that a transaction is in the final data phase if IRDY# is asserted, or that the bus is idle if IRDY# is negated.
GNT#[3:0] Grant	4	LOW	O	When asserted, GNT# indicates that the PCI master that asserted the corresponding REQ# has been granted bus control. When negated, it indicates that the corresponding PCI master must relinquish bus control at the end of the current transaction, or that the PCI master does not have bus control.
IRDY#	1	LOW	O	When asserted, IRDY# indicates that either write data is valid on AD[31:0] for a write transaction, or that the PCIC is ready to receive data on a PCIC read transaction.
PAR Parity	1	HIGH	O	When asserted, PAR reflects odd parity across the AD[31:0], and C / BE#[3:0] signals during both address and data phases, 1 cycle after the address or data phase. When negated, it reflects even parity.
			I	When asserted, PAR indicates odd parity driven by another master. When negated, it reflects even parity driven by another master.

Table 9-66 PCI Signal Listing *(Continued)*

SIGNAL NAME	PINS	ACTIVE	I/O	DESCRIPTION
PERR# Parity Error	1	LOW	O	When asserted, PERR# indicates that PCIC, as a target has detected a data parity error. When negated, it indicates that no error occurred.
			I	When asserted, it indicates that another target detected a data parity error while PCIC was master. When negated, it indicates that no error occurred.
REQ#[3:0] Request	4	LOW	I	When asserted, REQ#[3:0] indicates that an external master is requesting control of the PCI bus to run a transaction. When negated, it indicates that the external master does not have a transaction to run.
SERR# System Error	1	LOW	O	Indicates that a catastrophic error has occurred. This can be an address parity error, a data parity error during a special cycle, or an error response from the AFX bus when the PCIC is a target. When negated it indicates that no error occurred.
			I	When asserted it indicates another target has decoded a catastrophic error. When negated, it indicates that no error occurred.
STOP# Stop	1	LOW	O	When STOP# is asserted, the PCIC is requesting that the current bus master stop the transaction. When negated the current transaction can continue.
			I	When asserted, it indicates that a target is requesting PCIC to stop the current transaction. When negated the current transaction can continue.
TRDY# Target Ready	1	LOW	O	When asserted TRDY# indicates that PCIC, as a target, can complete the current data phase of the transaction. During a read, TRDY# indicates that valid data is on AD[31:0]. During a write it indicates the target is ready to accept data.
			I	When asserted, it indicates another target's ability to complete the current data phase of a transaction. When negated it indicates a wait from another target.

Table 9-67 PCI Bus Commands

C/BE#[3:0]	Command Type	Supported As Master	Supported As Slave	Definition
0000	Interrupt Acknowledge	yes	no	The Interrupt Acknowledge command is a read, implicitly addressed to the system interrupt controller.
0001	Special Cycle	yes	no	The Special Cycle command provides a simple message broadcast mechanism.
0010	I/O Read	yes	yes	The I/O Read command accesses devices mapped in I/O address space.
0011	I/O Write	yes	yes	The I/O Write command accesses devices mapped in I/O address space.
0100	reserved	no	no	
0101	reserved	no	no	
0110	Memory Read	yes	yes	The Memory Read command accesses devices mapped in the memory address space. The read when seen as a target fetches one 32 B line from memory when the address is so aligned.
0111	Memory Write	yes	yes	The Memory Write command accesses devices mapped in the memory address space.
1000	reserved	no	no	–
1001	reserved	no	no	–
1010	Configuration Read	yes	yes	The Configuration Read command is used to access the configuration space of each device, a device is selected when its IDSEL signals is asserted. (slave mode support when configured in Satellite mode only)
1011	Configuration Write	yes	yes	The Configuration Write command is used to access the configuration space of each device, a device is selected when its IDSEL signals is asserted. (slave mode support when configured in Satellite mode only)
1100	Memory Read Multiple	no	yes	The Memory Read Multiple command causes a prefetch of the next 32-B line. The PCIC treats this as a Memory Read command.
1101	Dual Access Cycle	no	no	This command is used to transfer 8-byte addresses to devices.
1110	Memory Read Line	no	yes	The Memory Read Line command is identical to the Memory Read command.
1111	Memory Write & Invalidate	no	yes	The Memory Write & Invalidate command is identical to the Memory Write command. No cache line function is supported.

9.11 PCI Protocol Fundamentals

Refer to the PCI specification (version 2.1) for a description of the PCI bus protocol.

9.11.1 PCI Addressing

PCI defines three physical address spaces: Memory, I/O, and configuration space. The memory and I/O spaces are standard. The configuration address space is defined to support a standardized method of configuring PCI devices, and is further defined by the PCI configuration space header. Each PCI device is responsible for its own address decoding. The microSPARC-IIep CPU can communicate with all three physical address spaces as a master, and responds to memory and I/O address spaces if enabled. The configuration registers of the microSPARC-IIep CPU are only available to the host (not through PCI configuration cycles) but the microSPARC-IIep CPU can read or write other configuration registers as a PCI master.

9.12 IAFX Bus Interface

9.12.1 IAFX Bus Overview

The AFX bus provides a memory-level interconnect protocol, originally defined between the microSPARC-II, system CPU, system memory, and a graphics subsystem. The AFX bus provided high bandwidth, low latency, and slave-only access to graphics by placing the graphics interface directly on the memory bus of a MicroSPARC-II-based system. The microSPARC-IIep CPU has an internal version of the AFX bus referred to as the IAFX bus. The PCIC uses the IAFX bus to provide a high-bandwidth, low latency bridge to the PCI Local Bus. The IAFX bus definition has been extended over that of the AFX bus to allow the PCIC to be a master on the IAFX bus, and to directly read and write main memory. This expanded protocol is described in Section 9.12.3, *DVMA (IAFX Master) Interface* on page 208 and has required some modifications from the microSPARC-II definition. The principal features of the IAFX Bus are:

- A 64-bit datapath
- A 28-bit physical address
- Synchronous operation at 1/3 CPU clock, except interrupts

- 1, 2, 4, or 8-byte transfers

New features for the PCIC IAFX Bus supported by MicroSPARC-IIep are:

- Internal Only operations—allows higher speed.
- Master access to main memory
- Direct data and address transfer from the PCIC (memory control from CPU).

Refer to the AFX (Local) bus specification in Appendix C for a detailed description of the AFX bus functionality. The PCIC implements separate DVMA (IAFX master) and PIO (IAFX slave) state machines. The DVMA (IAFX master) state machine controls IAFX transactions initiated from the PCIC (that is, PCI DVMA transactions). The master state machine controls the request logic for IAFX bus arbitration, as well as the sequencing logic for executing a master transaction. The PIO (IAFX slave) state machine controls the PCIC's internal logic when a CPU-initiated PIO (IAFX slave) transaction is decoded as targeted at the PCIC. The control includes return S_REPLY and P_REPLY handshakes on the IAFX bus, PIO (IAFX slave) buffer management, and PCI read buffer management.

9.12.2 IAFX Target Interface

This interface is the same as that of the normal AFX target (see Appendix C).

9.12.3 DVMA (IAFX Master) Interface

The PCIC bridge requires master access to main memory to support the PCI bus fully. The standard AFX bus definition does not support master access, as it is a slave-only bus protocol. The microSPARC-IIep CPU and the PCIC have implemented an extended AFX bus protocol, that defines six additional interface signals—and extends the definition of two existing pins—to support bus master operation by the PCIC. These signals allow the PCIC to read from and write to main memory directly. The PCIC does not implement a memory controller, but is slaved to the memory controller in the microSPARC-IIep CPU. When the PCIC initiates a master transaction, it signals the memory controller interface in the MicroSPARC-IIep to start the access as a random access, with the PCIC driving the address, and then to assert RAS and CAS in accordance with the memory operation currently executing.

Note – All signals used to support AFX and IAFX transfers are internal signals within the microSPARC-IIep CPU, interfacing only to the PCIC. They are described here for completeness, but are unavailable for external interfacing to the microSPARC-IIep CPU.

DVMA (IAFX Master) Signal Definitions are shown in *Table 9-68*.

Table 9-68 DVMA (IAFX Master) Signal Definition

Signal Name	QTY	DIR	Assert	Definition
RefCLK	1	I	H	RefCLK is the MicroSPARC-IIep reference clock. The memory subsystem and control operate at this clock frequency. All memory timing is synchronous with the rising edge of this clock, and all DVMA (IAFX Master) Signals are synchronous with it.
AM_READ	1	O	H	AM_READ is the DVMA (IAFX Master) Read / write control signal output by the PCIC. AM_READ is valid when AM_CSTB# is asserted. When AM_READ is active high, the PCIC is initiating a read operation. When AM_READ is low, the PCIC is initiating a write operation.
AM_BURST	1	O	H	AM_BURST is the DVMA (IAFX Master) row / column address control signal output by the PCIC. AM_BURST is valid when AM_CSTB# is asserted. When AM_BURST is active high, the address being driven by the PCIC is a row address (and implies a random access memory cycle being initiated). When AM_BURST is low, the address being driven by the PCIC is a column address (which implies a fast page-mode access).
AM_LOCK#	1	O	L	AM_LOCK# is the DVMA (IAFX Master) Lock signal output by the PCIC. This signal is used by the PCIC to retain control of the IAFX bus (and memory) in order to complete the current transaction properly. It is used to support sub-word write, implemented as read-modify-writes due to word parity, and also to support the PCI resource LOCK functionality.
AM_CSTB#	1	O	L	AM_CSTB# is the DVMA (IAFX Master) control strobe output by the PCIC. AM_CSTB# qualifies AM_READ and AM_BURST when asserted active low.
AM_AGNT#	1	I	L	AM_AGNT# is the DVMA (IAFX Master) address strobe output by the memory controller of microSPARC-IIep. When this strobe is active, it indicates that the PCIC can drive the IAFX address bus. (This signal is pipelined over one cycle to remain synchronous.)

9.12.3.1 DVMA (IAFX Master) Operations

The PCIC requires full memory access to support DMA masters on the PCI Local bus. However, the original MicroSPARC-II graphics local bus (AFX) was designed as a slave-only bus, not supporting external master access to memory. The microSPARC-IIep CPU, in conjunction with the PCIC, has six additional defined DVMA (IAFX master) signals to support external master memory access. These signals (prefixed with “AM” for IAFX master) allow the PCIC to request a cooperative memory operation, in which the microSPARC-IIep memory controller directs the memory control, and PCIC drives the memory address and data. In addition. the signal definition for the IAFX signals LO_ADDR and WRITE_L have been expanded. The LO_ADDR IAFX signal is defined to have a meaning during PCIC master operations, to reflect the need for either a row or column address

during a PCIC master memory operation, as directed by the microSPARC-IIep memory controller. The memory controller uses WRITE_L for different functions dependent on whether the master operation is a read or a write. During a master read, the WRITE_L signal is used as a data strobe, qualifying that the next cycle has valid data on the data bus (the one cycle pipeline is to allow for sampling the WRITE_L signal as a registered input). During a master write operation, WRITE_L is used to indicate when the current write operation has completed. The WRITE_L signal de-asserts (goes high) in the cycle before the last cycle of the write operation (the one cycle pipeline is present to allow for the input register).

A PCIC master memory operation begins with the PCIC asserting AM_CSTB#, the control strobe. This signal defines a memory operation request from PCIC, and must be asserted for each data transfer that PCIC masters (including the separate beats of data during a burst operation). The AM_CSTB# signal also qualifies the AM_READ, AM_BURST and AM_LOCK# signals, which are used to define the type of memory operation that the PCIC is initiating. The AM_READ signal defines the operation as either a read, when asserted (high), or a write when not asserted. The AM_BURST signal indicates if the operation is a random access or part of a burst sequence. Each master operation that asserts AM_CSTB# when AM_AGNT# (address grant) is not asserted, must start the operation as a random access (AM_BURST not asserted). The AM_BURST signal can only be asserted if it is part of a continuing operation (defined by AM_AGNT# being asserted when AM_CSTB# is asserted), which requires additional data transfers.

The microSPARC-IIep memory controller (referred to as the memory controller) responds to a PCIC master request by asserting AM_AGNT#. The AM_AGNT# grants PCIC ownership of the memory address bus (the PCIC drives the address bus in the next RefCLK cycle) and, for master writes, drives the data bus. Ownership of the memory address bus, signaled by AM_AGNT#, defines when PCIC is executing a master memory operation. The memory controller sequences the memory control signals, RAS, CAS, and WE appropriately to complete the operation. As each new PCIC master operation must be initiated as a random access, the memory controller always de-asserts RAS in parallel with asserting the AM_AGNT#. This action helps reduce latency by overlapping the RAS-precharge with the memory bus arbitration. The memory controller signals which address, row or column, the PCIC should be driving, by appropriately asserting / de-asserting the IAFX signal LO_ADDR. The LO_ADDR signal switches one cycle before PCIC should switch from driving the row address to driving the column address (the one cycle pipeline exists to allow for an input register on LO_ADDR). The memory controller then uses the WRITE_L data strobe to direct the PCIC to sample the data on the memory data bus during a master read operation, or signal the completion of a write during a master write operation. Once the memory controller deasserts the data strobe, it waits two RefCLK cycles for another assertion of AM_CSTB# to determine if another PCIC master memory operation is required. If AM_CSTB# is not asserted two RefCLK cycles after the deassertion of WRITE_L, the memory controller ends the PCIC master memory operation by de-asserting AM_AGNT#.

The PCIC can potentially burst a DMA write operation for a very long time, based on the PCI packer state machine packing separate PCI writes into a single long burst. (4 kilobytes is the theoretical maximum.) As this burst sequence may overlap with the refresh requirements, the memory controller must be able to interrupt any master burst sequence from the PCIC. The memory controller does this by removing AM_AGNT# (as a cycle is normally terminated) even if AM_CSTB# had been asserted within the 2 x RefCLK window after the de-assertion of AM_DSTB. The PCIC must recognize that the AM_CSTB# asserted was not acknowledged (by an assertion of AM_AGNT#), and retry the operation after a certain time-out period. (the current time-out period is 16 RefCLKs.)

The PCIC also supports the PCI exclusive access function by locking all of memory as a resource. This is performed by the PCIC asserting the AM_LOCK# signal during a AM_CSTB# master request. When AM_AGNT# is asserted, acknowledging the request, the memory controller locks out any access to memory other than by the PCIC. The memory controller sequences and completes the PCIC master memory operation normally when initiated with a AM_LOCK#. However, no other memory operation can occur (other than DRAM refresh) until the completion of a subsequent PCIC master memory operation that is initiated with AM_LOCK# de-asserted.

Rules and Relationships of DVMA (IAFX Master) Signals

The DVMA (IAFX Master) signals follow certain rules and have certain relationships that define their implementation. The following observations have no innate order, but are numbered for easy reference.

1. AM_CSTB# is both a request strobe from PCIC, as well as a qualifier for AM_READ, AM_BURST, and AM_LOCK#.

2. AM_AGNT# is the acknowledge and grant signal to the PCIC from the microSPARC-IIep memory controller. When asserted, AM_AGNT# indicates that the PCIC is driving the bus in the following cycle (pipelined one cycle to allow the corresponding output enables to be registered).

3. The first cycle in which AM_AGNT# is asserted is always a turnaround cycle (in which neither source drives the bus).

4. The cycle following the de-assertion of AM_AGNT# is always a turnaround cycle (in which neither source drives the bus).

5. The memory controller determines if it can service a PCIC request (AM_CSTB#) in the immediate state. If the memory controller cannot immediately service the request, it only retains the status that a request was made. The memory controller should hold-off all other operations at the

conclusion of the current operation. The memory controller does not retain any state of the last request from the PCIC, and therefore requires the PCIC to retry the same request.

6. During PCIC master writes, The PCIC must drive the data bus and the address bus simultaneously. When transitioning from a master read to a write in a burst, the PCIC drives the data bus on the third cycle following the assertion of AM_CSTB#. The second cycle following AM_CSTB# is a turnaround cycle. When transitioning from a master write to a burst read (an operation that is unlikely to be needed), the PCIC disables driving of the data bus the cycle following the assertion of AM_CSTB#. Following this, The memory controller drives the data bus on the second cycle following the assertion of AM_CSTB#.
7. For PCIC master reads, WRITE_L is asserted as the data strobe one cycle before the data is valid on the data bus.
8. For PCIC master writes, the assertion of WRITE_L is meaningless. The deassertion of WRITE_L indicates that the master write will complete in the following cycle. When another data transfer is pending, The PCIC uses WRITE_L to determine when to assert AM_CSTB#

Master Read Operations

The PCIC starts a master read operation by asserting AM_CSTB#, with AM_READ asserted, AM_BURST deasserted, and the condition of AM_LOCK# based on whether or not this is a PCI exclusive access. The PCIC then waits for the assertion of AM_AGNT# to determine when to begin the read operation. In the cycle following the assertion of AM_AGNT#, the PCIC drives the address bus with the row address based on LO_ADDR being de-asserted (assumed as this is the first access of a new operation). The memory controller then asserts LO_ADDR one cycle before the column address is required. the PCIC switches the address from row to column on the cycle following the asserting of LO_ADDR. The memory controller then asserts WRITE_L the cycle before the data is valid on the data bus. The PCIC samples the data on the data bus the cycle after the assertion of WRITE_L. If the PCIC has additional operations pending, it asserts AM_CSTB# two cycles following the assertion of WRITE_L. If no operations are pending, and therefore AM_CSTB# is not asserted in the two cycles following the assertion of WRITE_L, AM_AGNT# deasserts on the third cycle following the assertion of WRITE_L.

CHAPTER 10

Flash Memory Interface

The microSPARC-IIep flash memory interface provides a glueless connection to 28FxxxXX compatible flash memory devices. The interface has a programmable latency, which is set to 45 processor cycles per access on power up. After power up, this latency can be reprogrammed if required.

The flash interface is a word interface or a byte interface, and as such writes to the flash memory must be done as word writes or byte writes. There is no byte-collecting hardware to support the write operations. All writes are to the flash device as a memory mapped device.

10.1 Flash Memory Programming Interface

The flash memory or PCI address space can be selected as the boot memory. Refer to Section 11.7, *Boot Options* for selection of the boot address space. If the flash memory space is not selected in boot mode, it can still be accessed through the microSPARC-IIep address space mapping with PA[30:28]=0x2. (see *Table B-1* on page 251.)

The flash memory space resides in cacheable memory space within the microSPARC-IIep CPU, and subsequent references are satisfied from the cache. Boot mode accesses are non-cacheable while in boot mode. All load access widths are supported. For stores, however, only the native access width is supported. Bits 22:21 of the TLB replacement control register indicate the native access width. Values other than 0b01 indicate that the flash memory is 32 bits wide, while a value of 0b01 indicates that it has 8-bit width.

10.2 Flash Memory Speed

Refer to Section 5.8.5, *MID Register* on page 87 for more details on the flash ROM parameter setup. The flash memory access time is set as follows.

```
((flash memory speed) - 1) x 3 x CPU cycle time = flash memory access time
```

If the flash memory speed is set to 0x0 or 0x1 the flash memory access time used is 6 x CPU cycle time. These bits are readable and writable.

CHAPTER 11

Mode, Timing, and Test Controls

11.1 Overview

This section describes the functions:

- Reset logic (Section 11.2)
- Phase-locked loop (Section 11.3)
- Power management (Section 11.4)
- Clock control logic (Section 11.5)
- JTAG architecture (Section 11.6)
- Boot options (Section 11.7)

The JTAG logic controls all scan operation within the microSPARC-IIep CPU and in conjunction with the clock start/stop logic, enables the single step operation of the chip for debug purposes. All registers are scannable and are configured as one single scan chain for testing and debugging.

11.2 Reset Logic

11.2.1 General Reset and Watchdog Reset

When the reset input is active, the microSPARC-IIep CPU activates the PCI_RST# when operating in PCI host mode. In satellite mode, the PCI_RST# signal is an input pin and can reset the microSPARC-IIep CPU if enabled. See Section 9.9, *System Status and System Control* on page 201. All RAMs including the IU and FPU register files,

the data and instruction cache, and the TLB remain unchanged by the assertion of reset. On reset, state and pipeline registers internal to the IU are programmed to predetermined states. All other registers in the microSPARC-IIep CPU are reset to zero. See Section 9.9, *System Status and System Control*.

The microSPARC-IIep reset controller performs the simple task of driving the microSPARC-IIep CPU's internal reset lines, and inhibiting clocks during transitions on those lines to avoid timing violations as the flip-flops are reset.

The microSPARC-IIep CPU has two reset operations:

- General reset is triggered by:
  - Assertion of INPUT_RESET_L input pin on powerup and or any externally-triggered reset
  - Programmed software reset—see Section 9.9, *System Status and System Control*.
  - Assertion of PCI_RST# while in PCI satellite mode with reset enabled—see Section 9.9, *System Status and System Control*.

  During general reset, all registers except those in clock and reset logic and the TAP controller are reset. During scan-shift, INPUT_RESET_L is disabled to prevent loss of non-resettable state.
- Watchdog reset is triggered when the IU takes a trap and enters error state while the ET bit of the PSR is deasserted. However, the watchdog reset is delayed until no loads, stores, or instructions are in progress.

During transitions on the reset lines, the reset controller has another output that disables the outputs of the clock controller during transitions on the reset lines. This allows the heavily-loaded reset signals to propagate completely throughout the chip between clocks to avoid setup and hold time violations.

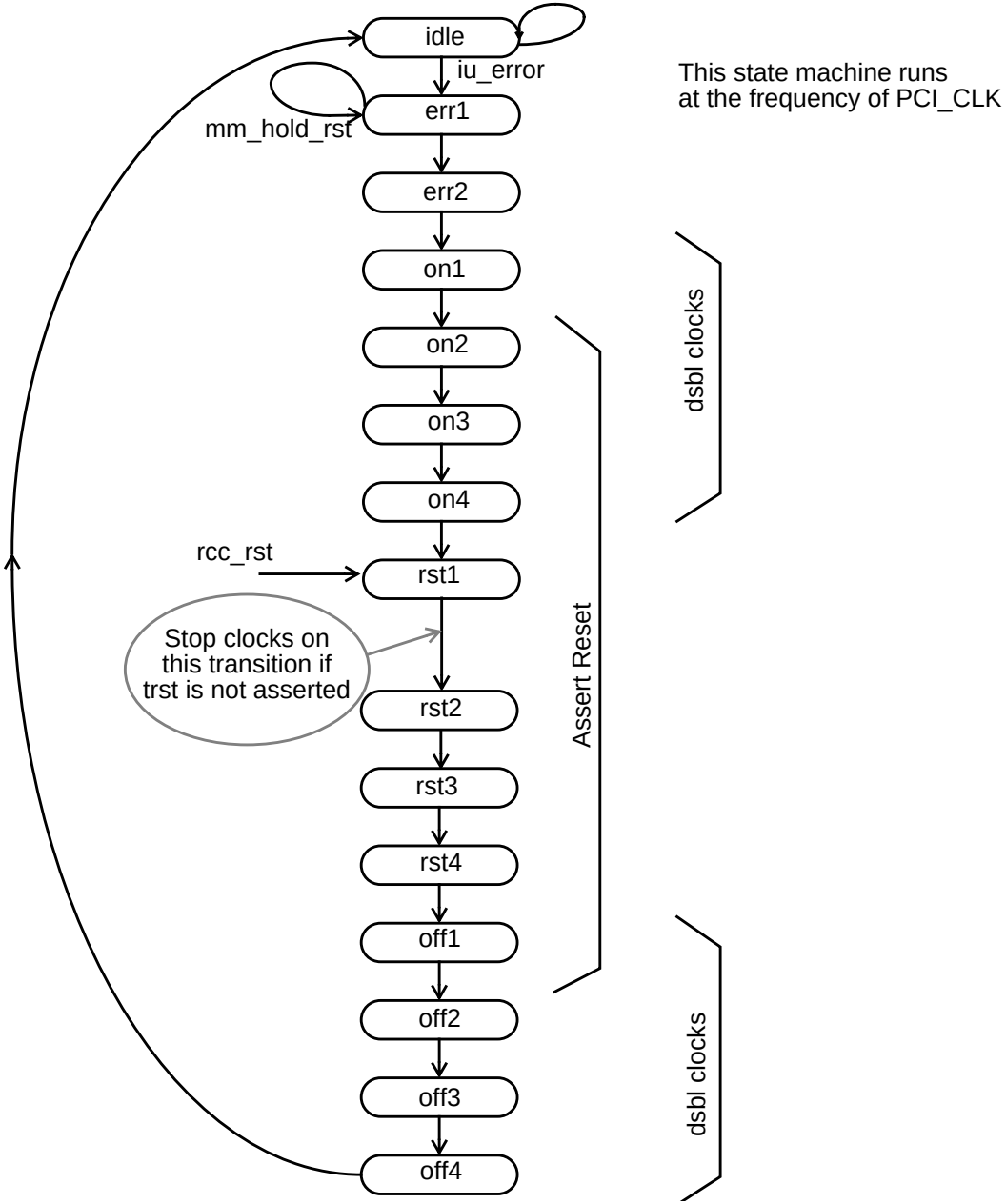


Figure 11-1 Reset State Machine

11.2.2 Reset Controller State Machine

The reset state machine is clocked at PCI_CLK. Assertion of RCC_RST synchronously resets the state machine into the rst1 state from any other state. The state machine thus stays in state rst1 for as long as RCC_RST is asserted. After completing a reset sequence, the state machine hangs in the idle state until either IU_ERROR or RCC_RST is asserted. If IU_ERROR is asserted while in the idle state, the state machine goes to state err1, waits there until MM_HOLD_RST is deasserted, and then completes the reset sequence and returns to idle. RESET_ANY and RESET_NONWD are asserted in states on2, on3, on4, rst1, rst2, rst3, rst4, and off1; if the reset sequence were initiated by IU_ERROR, only RESET_ANY is asserted; if initiated by RCC_RST, both RESET_ANY and RESET_NONWD are asserted.

Clocks are disabled in states on1, on2, on3, and on4 as the reset signal is turned on; they are disabled again in states off1, off2, off3, and off4 as reset is turned off again. This clock disabling does not put the clock state machine into the stopped state; it merely gates off the clock outputs. The reset lines are always deasserted during a clocks-disabled period, and for watchdog reset, they are asserted during a clocks-disabled period.

11.3 Phase-Locked Loop

The microSPARC-IIep CPU uses a phase-locked loop design to generate the internal high frequency clock. *Figure 11-2* shows the PLL block diagram.

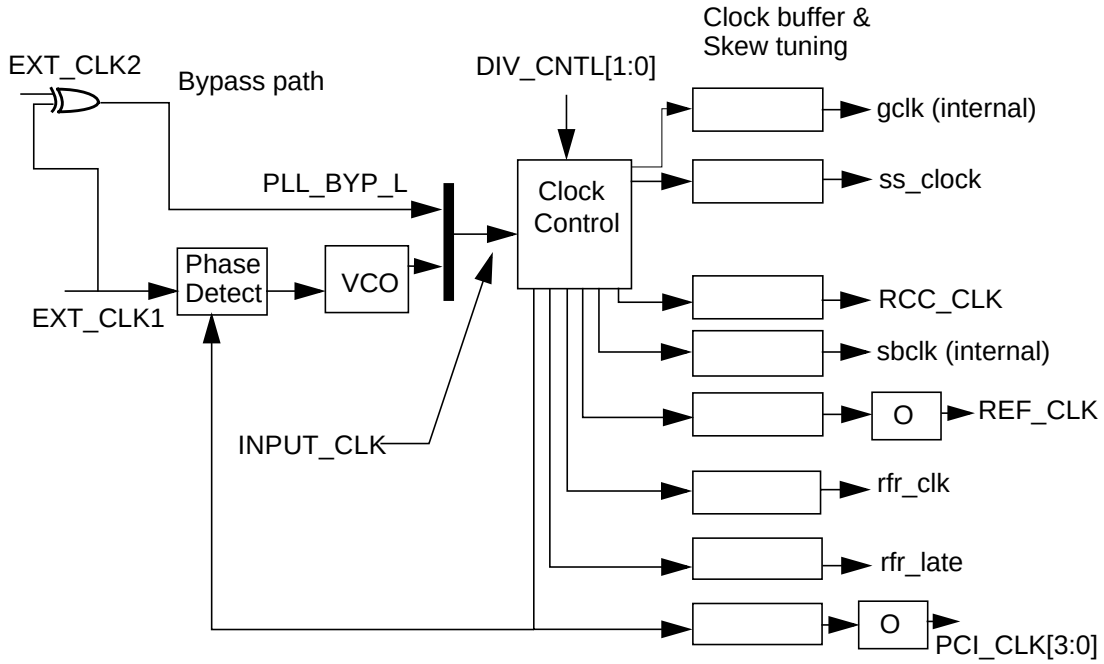


Figure 11-2 Phase-Locked Loop Block Diagram

The ss_clock (REF_CLOCK) is the 133 MHz system clock inside the microSPARC-IIep CPU. The PCI_CLK is the clock for some internal logic and state machines that do not require a high frequency. The PCI input clock is used in the PLL feedback loop such that the ss_clock is a multiple (3, 4, 5, or 6) of the PCI input clock.

The REF_CLK has the same frequency as the ss_clock and is routed off chip for testing purposes.

The voltage-controlled oscillator (VCO) generates a clock at twice the frequency of ss_clock. Depending on the state of PLL_BYP_L, different clock frequencies can be generated:

- PLL_BYP_L is asserted (i.e., tied to 0). EXT_CLK1 and EXT_CLK2 are XORed. If they are 90 degrees out of phase with each other, the clock generated from the XOR logic runs at twice the frequency of EXT_CLK1. When PLL_BYP_L is asserted, the microSPARC-IIep CPU operates in PCI host mode. (that is, the microSPARC-IIep CPU drives PCI_CLK[3:0] output pins.)
- PLL_BYP_L is deasserted. (tied to 1)

- If EXT_CLK2 is tied to 1 at power-up, the microSPARC-IIep CPU operates in PCI satellite mode (that is, an external PCI host droves the clock on the PCI bus). In that case, the PCI clock supplied by the external PCI host is connected to EXT_CLK1. The PCI_CLK[3:0] outputs of the microSPARC-IIep CPU are unconnected.
- If EXT_CLK2 is not tied to 1 at power-up, then the microSPARC-IIep CPU operates in PCI host mode, that is, the microSPARC-IIep CPU drives PCI_CLK[3:0] output pins. In that case the PCI_CLK[3:0] output pins carry the same frequency as that of EXT_CLK1.

DIV_CNTL_ is used to select the divider ratio for the PCI_CLK (3, 4, 5, or 6).

The following expression summarizes the clock generation:

```
input_clk = PLL_BYP_L?(2x * DIV_CNTL * EXT_CLK1 frequency):(EXT_CLK1 XOR EXT_CLK2)
```

ss_clock is at half the frequency of input_clk.

Clock skew between ss_clock and PCI_CLK, ss_clock and REF_CLK should be less than 1 ns. The PLL is designed to operate at up to 400 megahertz.

11.4 Power Management

The following paragraphs list the microSPARC-IIep power management features.

Cache RAM powerdown — Whenever the cache controllers detect that one of the cache RAMs need not be accessed in a given clock cycle, that RAM is automatically put into powerdown mode for that cycle. In this mode the RAM consumes minimal power. This mode is used when the cache is disabled, when the CPU is waiting for cache miss data to be returned from memory, or when the chip is in standby mode.

- The microSPARC-IIep CPU includes a programmable bit in the MID register that allows the processor to enter the power down mode internally, without the need of an external monitor (see Section 5.8.3, *Memory Fault Status Register*). When the processor sets the standby bit in the MID register, all internal operations are allowed to complete, and if there is no activity on the PCI bus, the processor shuts down the internal clocks and enters standby mode. While in standby, the processor parks the PCI bus at itself if it is operating in the PCI host mode. Any request to use the PCI bus or any interrupt activity (counters included) resets the bit of the MID register and takes the processor out of standby mode.

  DMA activity on the PCI bus takes the processor out of standby state by resetting the MID register bit even though the processor is not involved. In order for the processor to return to standby, the bit in the MID register must be set again (that is, there must be an idle loop).

- Self-refresh DRAM mode — In this mode, the DRAMs operate in self-refresh mode, assuming that the DRAMs have self-refresh capability. It is controlled by bit 13 of the Processor Control register PCR. After a 1 is written to PCR[13], the DRAMs enter self-refresh mode within 2 µs (see Section 5.7.1, *Processor Control Register* on page 73).

11.5 Clock Control Logic

The microSPARC-IIep clock controller generates the clock signals used by all of the microSPARC-IIep CPU (except the TAP controller) as well as the PCI_CLK[3:0]. PCI_CLK[3:0] drive external PCI devices when the microSPARC-IIep CPU is operating in PCI host mode. Otherwise, these output pins are unconnected and clocks for PCI devices are supplied by the external PCI host. Its operation is controlled by the clock control register (CCR), a collection of internal register bits that is writable only by JTAG signals. On reset, the CCR is cleared. Subsequent scan-shift operations can be used to set bits of the CCR and alter the operation of the clock state machine as described in this section.

The microSPARC-IIep clock controller is designed to interface to a simple internal cycle counter (ICC) for precise, at-speed control of system clocking. The ICC is a simple binary counter that increments on rising edges of PCI_CLK.

Note – The ICC is currently not accessible via scan or JTAG.

The interface consists of three microSPARC-IIep I/O pins:

- PCI_CLKn — A clock output. This output is used to clock some external logic as well as the ICC.
- ext_event (input) — This input is immediately registered in a PCI_CLK-clocked flip-flop. Under control of some clock control register (CCR) bits, a logic 1 in this flip-flop causes clocks to stop either at the next RCC_CLK edge or the next PCI_CLK edge. This input should be driven by the terminal_count output of the ICC and perhaps ORed with other externally-detected clock-stop signals. In a standard binary up-counter, the terminal count output is asserted when the counter contains all ones (i.e., a two's-complement value of -1).
- int_event (output) — This is the output of a PCI_CLK-clocked flip-flop. It is asserted whenever an internally-detected event occurs (e.g., virtual address match). These events can, under control of some CCR bits, stop clocks; however, whether or not they stop clocks, they always cause assertion of the int_event output. This output can be used to trigger a logic analyzer; in addition, it can be used in conjunction with the ICC as described in Section 11.5.7, *Stop Clocks N Cycles after Internal Event*.

In addition, there are two microSPARC-IIep input pins that control the internal clock divider: these bits specify the (RCC_CLK: PCI_CLK) frequency ratio D (see *Table 11-1*).

Table 11-1 Internal Clock Divide Control

DIV_CTL[1:0]		RCC_CLK Range	
	D (Clock Divide)	RCC_CLK (MHz)	PHI[2:0]
01	3	100	0,1,2
10	4	133	0,1,2,3
11	5	166	0,1,2,3,4
00	6	200	0,1,2,3,4,5

The RCC_CLK range shown in *Table 11-1* is the range of internal RCC_CLK frequencies that is obtained when PCI_CLK spans its legal range up to 33 MHz. The PHI[2:0] column shows the sequence of states traversed by the PHI[2:0] field of the CCR in each PCI_CLK cycle. PHI[2:0] transitions to the next state in the sequence on each RCC_CLK rising edge and the RCC_CLK rising edge that coincides with the PCI_CLK rising edge always causes PHI[2:0] to transition to the 0 state.

The ICC / microSPARC-IIep interface runs at the PCI_CLK clock rate and the signal I/O connects directly to inputs or outputs of flip-flops within the microSPARC-IIep CPU. Thus, the ICC logic has nearly a full PCI cycle in which to set up its output to the ext_event input.

11.5.1 Stopping Clocks

Rather than using the ICC., clocks can be started with the CCR bit.

11.5.2 Starting Clocks

Rather than using the ICC., clocks can be started with the CCR bit.

11.5.3 Single-Step

This mode does not require the use of the ICC. From a clock-stopped state, set both the stop_clocks and start bits of the CCR. A single active-low RCC_CLK pulse is issued, with a pulse width of 1/2 the normal RCC_CLK period; if the RCC_CLK

pulse causes PHI[2:0] to transition to the 0 state, a single active-low PCI_CLK pulse is also issued. Its pulse width is 1/2 the normal PCI_CLK period and its rising edge coincides with the rising edge of RCC_CLK.

Figure 11-3 shows a divide-by-three example.

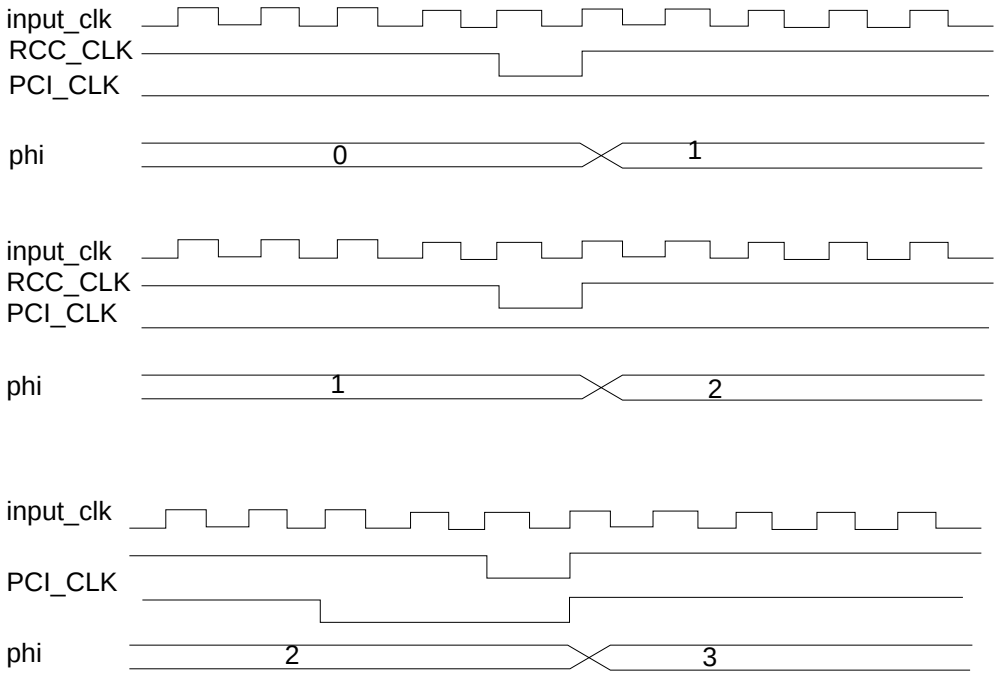


Figure 11-3 Divide-by-3 Example

11.5.4 Counting Clocks

When the ICC is enabled, it increments on every rising edge of PIC_CLK[3:0]. Since the states of the ICC and the CCR are accessible using scan, the number of clocks issued between any two points in time can be calculated by scanning out the state information before clocks are started and again after they have been stopped. The following formula can be used.

```
N = D*(ICC.after-ICC.before) + (phi.after-phi.before)
```

- D is the divider ratio (3, 4, 5, or 6) specified by DIV_CTL[1:0].

- ICC.before and ICC.after are the respective values of the external clock counter before and after clocks have been issued.
- phi.before and phi.after are the corresponding values of the phi[2:0] bits of the CCR.

This formula assumes that the ICC has not wrapped; the ICC control logic should contain a wraparound detector that can be read by scan.

11.5.5 Issuing N Clocks

The ICC can be used to issue exactly N system clocks, at full speed. N can be any number from 1 to approximately $D(2^X)$, where D is the (RCC_CLK: PCI_CLK) clock divide ratio and X is the number of bits in the ICC; for example, a 32-bit ICC allows control of clocks over a 200-second range at 80-MHz operation in a divide-by-4 mode. This function does not require the use of the int_event output.

To issue N clocks from a clocks-stopped state, several CCR fields, as well as the ICC register, are involved. Scan a 1 into the start and stop_on_ext_event control bits, copy (using scan) the current phi[2:0] field into the ref_phi[2:0] field, and scan appropriate values into the extra_cycles[2:0] field and into the ICC. The number of clocks issued is given by this formula:

```
N = D*(-ICC.before) + extra_cycles + 1;
```

where -ICC.before is the positive number formed by taking the two's-complement of the scanned-in ICC value. Thus, to issue N clocks, scan the twos-complement of (N-1)/D into the ICC, and scan (N-1)%D into extra_cycles[2:0], where "/" is integer divide with the remainder discarded, and '%' is the remainder of integer divide. For example, to issue 17 clocks in divide-by-3 mode, scan -((17-1)/3) = 0xFFFFFFFB into the ICC, and (17-1)%3 = 1 into extra_cycles[2:0].

Because the value scanned into the ICC is treated as a negative number to be counted up towards zero, the formula above works only when (N-1)/D > 0, i.e. when (N > D). For (0 < N ≤ D), scan 00000000 into the ICC, scan 1 into the ext_event_sb1 bit of the CCR, and scan (N-1)%D into extra_cycles[2:0].

A complete algorithm is shown below. It includes a few other CCR bits that must be set to specific states.

```
if (N < 1)
  error;
else {
  ICC = -(N-1)/D;
  CCR.extra_cycles = (N-1)%D;
  CCR.ref_phi = CCR.phi;
  if (N ≤ D)
```

```
        CCR.ext_event_sb1 = 1;
    else
        CCR.ext_event_sb1 = 0;
    CCR.start = 1; CCR.stop_on_ext_event = 1;
    CCR.stop_int_to_ext = 0;
    CCR.int_to_ext = 0;
    CCR.ext_event_sb2 = 0;
}
```

11.5.6 Stop Clocks on Internal Event

This facility does not require the use of the ICC. To stop clocks on detection of an internal event, set the stop_on_int_event bit of the CCR and enable the desired internal event detection logic. Clocks, with some limitations, stop at the end of the RCC_CLK cycle in which the input to the int_event flip-flop is asserted. The limitation of this mode is that clocks cannot stop in phi==2 when D==3, phi==3 when D==4, or in phi==3 or 4 when D==5;. If an internal event occurs in either of these situations, clocks stop one cycle later (i.e., in phi==0). Note that, since the int_event flip-flop is clocked only on PCI_CLK edges, the int_event output pin is not set by the internal event that stops the clocks, unless clocks have stopped in phi==0.

11.5.7 Stop Clocks N Cycles after Internal Event

In this mode, the ICC is held until an internal event occurs. The internal event does not stop clocks, but causes assertion of the int_event output. The int_event output remains asserted until it is cleared by scan. The ICC is enabled to count whenever int_event is asserted, so clocks continue to run until ext_event is asserted, either by ICC or by another external event detector. The intent of this mode is to issue exactly N more clocks than would have been issued in stop_on_int_event mode (see Section 11.5.6) That is, exactly N clocks are issued after the first rcc_clock positive edge at which the input to the int_event flip-flop is asserted. Logic in the clock controller records the clock phase in which the internal event occurred, and this information is factored into the subsequent clock stop on external event, so that N can be any integer. Due to timing limitations, N must be greater than D.

To support this mode, the ICC must have logic which, under scan control, holds the count when int_event is not asserted.

To have clocks continue for exactly N cycles after the cycle in which the internal event occurs, several CCR fields, as well as the ICC register, are involved. A 1 must be scanned into the start and int_to_ext CCR bits, and a 0 scanned into the

stop_on_ext_event and stop_int_to_ext CCR bits. Scan appropriate values into the extra_cycles[2:0] field and into the ICC. The following formula gives the number of additional clocks to be issued after the cycle in which the internal event occurs.

```
N = D*(-ICC.before) + extra_cycles + D;
```

where -ICC.before is the positive number that results from taking the two's-complement of the scanned-in ICC value. Thus, to issue N clocks, scan the two's complement of (N / D - 1) into the ICC, and scan (N%D) into extra_cycles[2:0], where "/" is integer divide with the remainder discarded, and '%' is the remainder of integer divide. For example, to issue 35 clocks after an internal event in divide-by-4 mode, scan -(35 / 4 - 1) = 0xfffffff9 into the ICC, and (35%4) = 3 into extra_cycles[2:0]. As described for stop_on_ext_event mode, if the formula gives an initial ICC value of 0, also scan a 1 into ext_event_sb1.

A complete algorithm:

```
if (N ≤ D)
  error;
else {
  ICC = -(N/D - 1);
  CCR.extra_cycles = (N%D);
  CCR.ref_phi = CCR.phi;
  CCR.start = 1; CCR.int_to_ext = 1;
  CCR.stop_on_ext_event = 0;
  CCR.stop_int_to_ext = 0;
  CCR.int_event = 0;
  if (N < (2*D))
      CCR.ext_event_sb1 = 1;
  else
      CCR.ext_event_sb1 = 0;
  CCR.ext_event_sb2 = 0;
}
```

11.5.8 Stop Clocks after N Internal Events

In this mode clocks are stopped after the Nth detected internal event. Clocks are stopped as described in Section 11.5.7, *Stop Clocks N Cycles after Internal Event* for stop_on_int_event mode, except that the first (N-1) PCI_CLK cycles of int_event assertion are ignored. Due to the limited resolution of the ICC interface, if more than one internal event occurs within a single PCI_CLK cycle, that counts as only a single event.

This mode is enabled by the stop_nth_event CCR bit, and ICC needs a scannable control bit that enables it to count only while int_event is active.

To use this mode, load ICC with (2-N) and turn on stop_nth_event. As with other modes described above, some special action is required if the initial ICC value given by this formula is non-negative. There follows a complete algorithm:

```
if (N < 1)
  error;
else {
  ICC = (2 - N);
  CCR.start = 1;
  CCR.int_to_ext = 0;
  CCR.stop_on_ext_event = 1;
  CCR.stop_int_to_ext = 0;
  CCR.int_event = 0; if (N == 1)
  CCR.ext_event_sb2 = 1; else
  CCR.ext_event_sb2 = 0;
  if (N == 2)
      CCR.ext_event_sb1 = 1;
  else
      CCR.ext_event_sb1 = 0;
}
```

11.5.9 Clock Control Register (CCR) Bits

A list of the clock control register bits follows. These bits are only accessible by scan, and their function is described above.

- start
- stop_clocks
- stop_on_int_event
- stop_on_ext_event
- stop_int_to_ext
- stop_nth_event
- extra_cycles[2:0]
- int_event
- ext_event_sb1
- ext_event_sb2
- phi[2:0] (Treat this as read only)
- ref_phi[2:0]

11.6 JTAG Architecture

A variety of microSPARC-IIep test and diagnostic functions, including internal scan, boundary scan, and clock control are controlled through an IEEE 1149.1 (JTAG) standard test access port (TAP). Commands and data are sent as serial data between the JTAG master and the microSPARC-IIep chip (a JTAG slave), through a 4-wire serial testability bus (JTAG bus). The TAP interface to the JTAG bus uses five dedicated pins on the microSPARC-IIep chip. These pins are

- TCK - input - test clock
- TMS - input - test mode select
- TDI - input - test data input
- TRST_L - input - JTAG TAP reset (asynchronous)
- TDO - output - test data output

For more details on the IEEE protocol, refer to the IEEE document *IEEE Standard Test Access Port and Boundary-Scan Architecture*, published by IEEE—see *Bibliography* on page 275.

11.6.1 Board Level Architecture

Any microSPARC-IIep-based system contains several JTAG-compatible chips. These are connected using the minimum (single TMS signal) configuration as described in the 1149.1 specification (Figure 3-1, IEEE 1149.1 standards manual). This configuration contains three broadcast signals (TMS, TCK, and TRST,) that are fed from the JTAG master to all JTAG slaves in parallel, and also to a fed to a serial path formed by a daisy-chain connection of the serial test data pins (TDI and TDO) of all slaves.

The TAP supports a BYPASS instruction which places a minimum shift path (1 bit) between the chip's TDI and TDO pins. This arrangement allows efficient access to any single chip in the daisy-chain without board-level multiplexing.

11.6.2 Test Access Port (TAP)

The TAP consists of a TAP controller, a number of shift registers including an instruction register (IR), and multiple data registers.

The TAP controller is a synchronous finite state machine that controls the sequence of operations of the JTAG test circuitry in response to changes at the JTAG bus—specifically, in response to changes at the TMS input with respect to the TCK input.

Note – The TAP controller is asynchronous with respect to the system clock(s), and can therefore be used to control the clock control logic.

The TAP finite state machine (FSM) implements 16 states as required by the 1149.1 protocol.

The IR is a 6-bit register that allows a test instruction to be shifted into the microSPARC-IIep CPU. The instruction selects the test to be performed and the test data register to be accessed. The supported instructions are listed in Section 11.6.3, *JTAG Instructions*.

Although any number of loops may be supported by the TAP, the finite state machine in the TAP controller only distinguishes between the IR and a data register. The specific data register can be decoded from the instruction in the IR.

The following data registers are supported in the microSPARC-IIep TAP.

- Bypass register — A single-bit shift register for efficient board-level scan
- Device I.D. register — A 32-bit register with the field shown in *Figure 11-4*

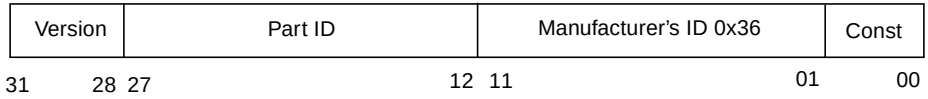


Figure 11-4 Device ID Register Contents

Field Definitions:

- [31:28]: Version — represent the version number, 0x1 for this version
- [27:12]: Part ID — representative part number as assigned by Vendor, 0x016d
- [11:01]: Manufacturer's ID — representative manufacturer's ID as per JEDEC, 0x36
- [0]: Const — Tied to a constant logic '1'
- Value in ID Register: 32'h 00000009
- Data register — this is a two-bit clock control register to sample outputs from the clock controller (CCR)
- Boundary Scan Register — single scan chain consisting of all of the boundary scan cells (input, output and in/out cells)
- MISC block Internal Scan Registers - a single scan chain of all the internal scan flipflops in the MISC block

11.6.3 JTAG Instructions

The instructions listed in *Table 11-2* are supported by the microSPARC-IIep TAP. The table contains the instruction bit-values and mnemonics, with the data register selected by each instruction.

Table 11-2 JTAG Instructions

Value	Name of Instruction	Data Register	Scan Chains Accessed
000000[1]	EXTEST	Boundary Scan register	boundary scan chain
000001[1]	SAMPLE	Boundary Scan register	boundary scan chain
000010	INTEST	Boundary Scan register	boundary scan chain
000011	ATEINTEST	Boundary Scan register	boundary scan chain
100000	IDCODE	JTAG ID register	ID register scan chain
111111[1]	BYPASS	Bypass register	Bypass register
011110	SEL_CCR	Clock Control register	Clock Control register chain
110000	CLD_RST	Bypass register	Bypass register

1. Encodings fixed by IEEE JTAG protocol mappings.

Notes:

- The two internal scan chain instructions differ with respect to the scan chain clocking during CAPTURE_DR state of the TAP FSM. sel_int_scan is used for ATPG tests, where a clock pulse is needed to capture the next state when the scan_mode signal is in the inactive state between shift cycles. The other scan instruction, sel_dbg_scan is used during debug to read and write the scan chain. No pulse is generated during the transition from “shift → capture → shift” states. In other words, the scan state is preserved during the shift, capture, shift cycle. cycle. Only the internal scan chain within the MISC block is accessible. All core logic scan chains are directly accessible by means of four dual-use, scan-input pins and four dual-use, scan-output pins.
- The TDO output becomes valid at the falling edge of TCK according to the 1149.1 specification. The TDI input is clocked during the rising edge of TCK.
- The ATEINTEST operation is used to load the boundary scan flip-flops after which, if it enters the run_test_idle state, the JTAG controller generates a single TCK pulse.
- Although the capability exists to single step the chip through another mechanism, using sys_clock itself, the ATEINTEST option provides the capability to perform ICT on the ATE at a slow speed.

- SEL_CCR is used to sample two bits (stopped) from the clock controller block. These two bits are synchronized—using a 2-stage synchronizer using TCK—before being sampled during the shift-DR state.

11.6.4 JTAG Interface to MISC

The JTAG block provides two key signals to the clock controller section, two signals directly to the microSPARC-IIep core, and a five-wire control signal to the boundary scan flip-flops.

11.6.4.1 Clock Controller Interface

Signals testclk and testclken are generated in the JTAG block and sent to the clock controller.

testclken is an active high signal that switches the 100 MHz ss_clock to the core. This 100 MHz clock is taken from the normal 100 MHz clock signal to testclk. This happens only for certain JTAG instructions. They are:

SEL_INT_SCAN, SEL_DBG_SCAN, INTEST, ATEINTEST

For the remaining instructions: extest, sample, bypass, idcode, and sel_ccr, testclken remains inactive to enable the normal 70 MHz clock to the microSPARC-IIep core. The testclken signal is synchronized inside the clock controller using PCI_CLK. By design, testclken is activated at least three TCK cycles before the testclk signal becomes active. testclken signal changes state only during the transition from exit1-IR state of the instruction scan cycle.

testclk is a gated version of TCK processed with the gating signals: sel_instruction, shift (function of shift_DR), and capture (capture-DR) states.

11.6.4.2 Boundary Control Interface

The five-wire boundary control signal corresponds to: bin_cap, bout_cap, b_sen, b_uen, b_mode.

Signals bin_cap and bout_cap are generated during the capture-DR state and are used to load the values on the pins or the outputs of the core into the boundary scan flip-flop. To avoid race conditions, b_sen is generated on the falling edge of the tck and is used as a scan_en signal for the boundary scan flip-flop. b_uen is an update signal for the boundary scan update latch that occurs at the falling edge of TCK.

b_mode is a mux control signal that selects between the direct pin input and the value in the update latch. This signal changes both during the update-IR state and when the tap goes back to test-logic-reset state on the falling edge of TCK.

11.6.4.3 RESET Mechanism

In its active low condition, the independent TRST_L signal sets the TAP into the tap_logic_reset state. This signal asynchronously sets the TAP state machine to the tap_logic_reset state. It adheres to the 1149.1 IEEE protocol with respect to the initialization through reset mechanism. There is no minimum active time requirement on this reset signal. If the board has no extra oscillator for TCK, the JTAG reset pin (TRST_L) can be tied to an active low signal to disable JTAG operations in the chip.

TDI and TMS inputs have pullups on the pad and consequently show input ones if not connected. With a free running TCK, the TAP enters the tap_logic_reset state at the end of five TCKs.

11.6.5 JTAG Operation

The following description covers some of the basic operations that, when combined, enable the user to run any of the JTAG instructions specified above. They are given here for an understanding of the TAP state transitions during various JTAG operations.

The JTAG I/O consists of inputs: TCK, TMS, TDI, TRST, and output: TDO, all of which are chip I/O. The other inputs to the chip are either in a *don't care* state or a predetermined state and should not affect the operation of the JTAG controller. Note that, for a robust operation of the chip, this procedure should ensure that the system is reset on entering, leaving, and returning to JTAG operations. Once in the tap_logic_reset state, all JTAG outputs become inactive and the chip should be in normal functional mode.

The tap state encodings, in hexadecimal, are shown in *Table 11-3*.

Table 11-3 Tap State Encodings

Encoding (hex.)	TAP State
0	exit2-DR
1	exit1-DR
2	shift-DR
3	pause-DR

Table 11-3 Tap State Encodings *(Continued)*

Encoding (hex.)	TAP State
4	select-IR
5	update-DR
6	capture-DR
7	select-DR
8	exit2-IR
9	exit1-IR
A	shift-IR
B	pause-IR
C	run-test-idle
D	update-IR
E	capture-IR
F	test-logic-reset

To run the JTAG instructions, the following TAP state traversal is done for the various subtasks:

Instruction Scan

The Instruction Scan Sequence is shown in *Table 11-4*.

Table 11-4 Instruction Scan Sequence

Code	TAP State	Comments
F	test-logic-reset	
C	run-test-idle	
7	select-DR	
4	select-IR	
E	capture-IR	
9	exit1-IR	
B	pause-IR	

Table 11-4 Instruction Scan Sequence

Code	TAP State	Comments
8	exit2-IR	
A	shift-IR	for 6 clocks; opcode is shifted through tdi while in this state
9	exit1-IR	

Data Scan

The Data Scan sequence is shown in *Table 11-5*.

Table 11-5 Data Scan Sequence

Code	TAP State	Comments
9	exit1-IR	
B	pause-IR	
8	exit2-IR	
D	update-IR	
C	run-test-idle	
7	select-DR	
6	capture-DR	
1	exit1-DR	
3	pause-DR	
0	exit2-DR	
2	shift-DR	# of shifts equal to length of scan chain
1	exit1-DR	

At state “D” the decode instruction is latched on the falling edge of TCK. Data is shifted into the appropriate data register during the shift cycle. At the end of shift, the data moves to the exit-DR(1) state.

Return to New Instruction

The Data Scan sequence is shown in *Table 11-6*.

Table 11-6 Data Scan sequence

Code	TAP State	Comments
2	shift-DR	
1	exit1-DR	
3	pause-DR	
0	exit2-DR	
5	update-DR	
C	run-test-idle	wait in this state and go back to Instruction Scan step

Figure 11-5 shows the JTAG logic block diagram. *Figure 11-6* shows the JTAG data and instruction registers.

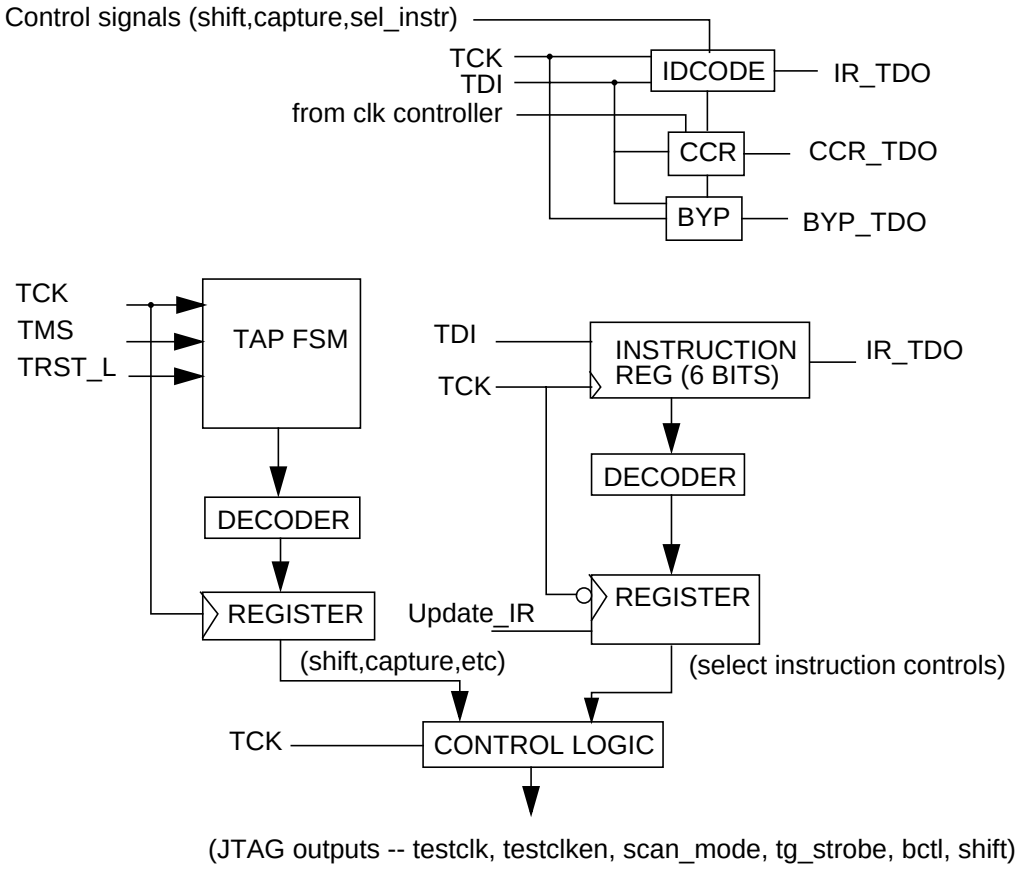


Figure 11-5 JTAG Logic Block Diagram

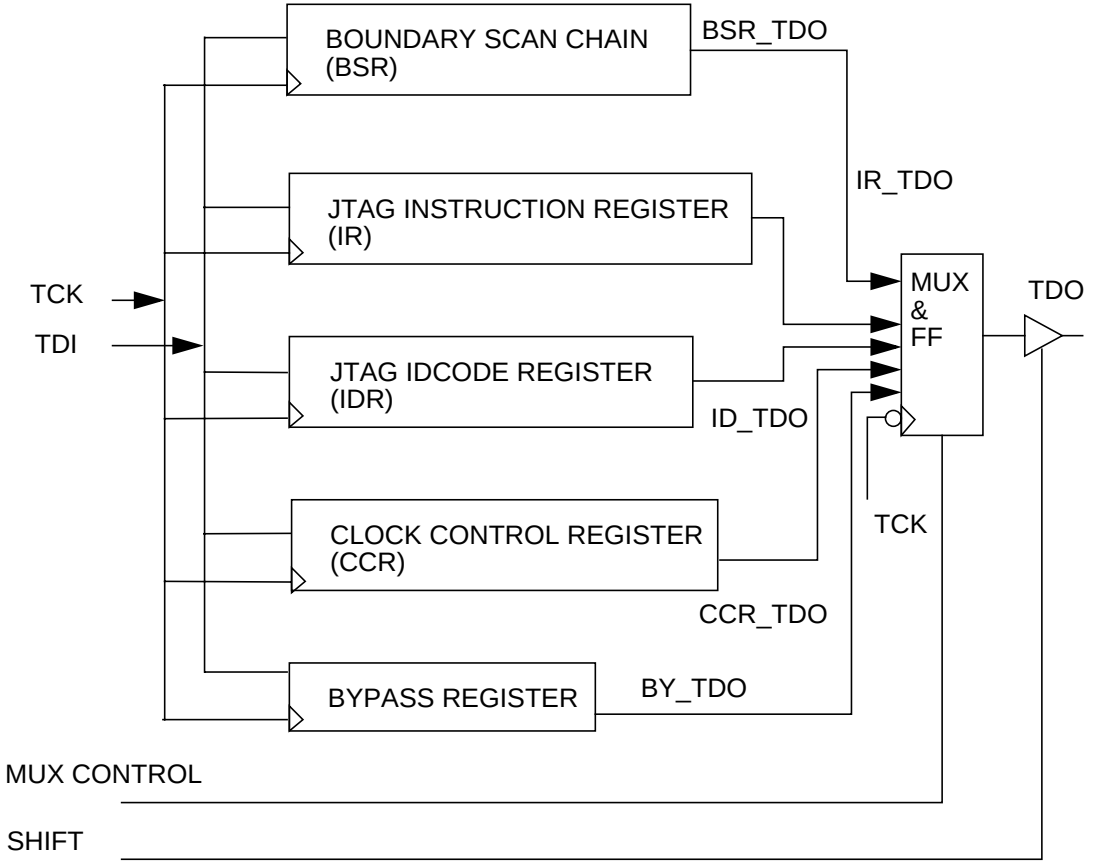


Figure 11-6 JTAG Data & Instruction Registers

11.6.6 CLK_RST TAP Instruction

The microSPARC-IIep clk_cntl block is a collection of non-scanned logic that generates the various clock waveforms used both on and off the microSPARC-IIep chip. Although this logic is not directly scannable, the microSPARC-IIep CPU implements a private TAP instruction for initializing the state of the flip-flops in the clk_cntl block. This instruction is intended for use by a tester, since it requires precise control of the waveforms driven onto the EXT_CLK1/EXT_CLK2 microSPARC-IIep input pins.

The instruction mnemonic is CLK_RST, and its binary opcode is 110000. Its behavior is identical to that of the BYPASS instruction, except that the internal signal clk_rst_l is asserted whenever the CLK_RST opcode appears on the TAP instruction register

output latch; that is, starting at the falling edge of JTAG_CK when the TAP state machine is in the update-ir state — see IEEE Std 1149.1 for details of the TAP state machine operation. While clk_rst_l is asserted, some of the flip-flops in clk_cntl are synchronously reset at the rising edge of the high-speed input_clock.

It is intended that the CLK_RST operation—see *Figure 11-7*—be used only when the microSPARC-IIep PLL_BYP_L input pin is driven to 0, that is when the internal phase-locked-loop is bypassed. In that mode, input_clock is equal to the XOR of the EXT_CLK1 and EXT_CLK2 input pins. clk_cntl can be reset to a known state using the algorithm:

1. Apply clocks to JTAG_CK, drive JTAG_TDI=1, and drive PLL_BYP_L=0 for the duration of the test. Drive EXT_CLK1=0 and EXT_CLK2=0 through step 4 below, with the exception of a single 0 → 1 → 0 pulse on EXT_CLK1 in step 4.

2. Assert JTAG_TRST_L, then de-assert it, to reset the TAP controller.

3. Apply this sequence of values to jtag_ms, applying a new value at each negative edge of jtag_ck (the number below each value is a cycle count, for reference):

Value	1...	1	0	1	1	0	0	0	1	1	1	1	1...
Cycle count		0	1	2	3	4	5	6	7	8	9	10	11

   Note that in cycle 5, the IR is parallel-loaded with 000001. In cycle 6 and 7, ones are shifted into the MSB end of the IR. The result is a 110000 in the IR.

4. In cycle 10 of the sequence above, apply a single 0 → 1 → 0 pulse to EXT_CLK1. The rising edge of this pulse resets the clk_cntl block.

5. After cycle 11 of the sequence above, clk_cntl is reset and the TAP controller is in the test-logic-reset state. JTAG_TRST can now be asserted and clocks applied to EXT_CLK1 and EXT_CLK2 to start the test.

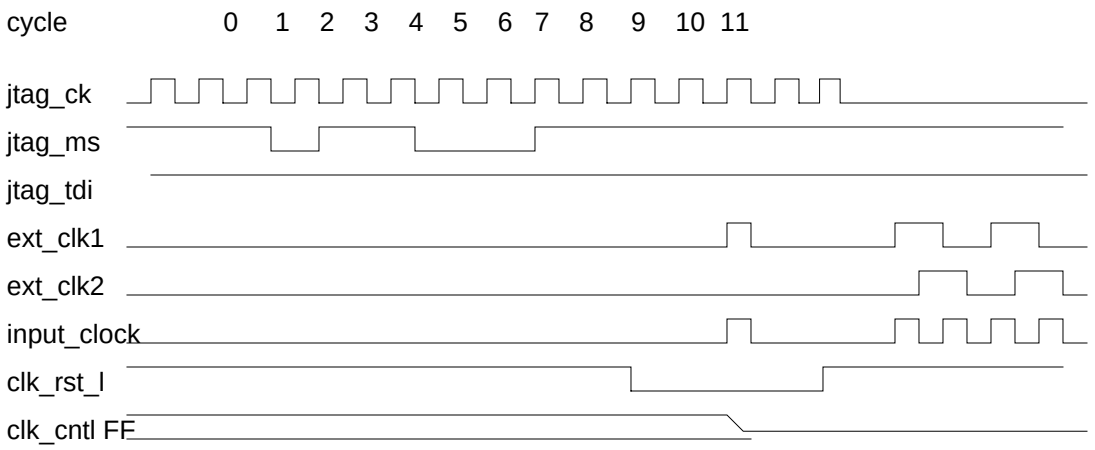


Figure 11-7 JTAG Clk Reset Operation

11.7 Boot Options

The microSPARC-IIep CPU provides four boot options (see *Table 11-7*). The options are set using the BM_SEL[1:0] pins that are programmer-visible in bits 22:21 of the TLB replacement control register.

Table 11-7 Boot Mode Select (BM_SEL)

BM_SEL[1:0]	Boot From:
00	32-Bit flash memory on memory data bus (cacheable)
01	8-Bit flash memory on memory data bus (cacheable)
10	PCI Bus, boot from addresses in range 0xF000.000–0xF0FF.FFFF (non-cacheable)
11	PCI Bus, boot from addresses in range 0xFFFF.000–0xFFFF.FFFF (non-cacheable)

BM_SEL options 00, 01 are described in Chapter 10, *Flash Memory Interface*.

Selection of options 10, 11 causes interception by the PCIC.

- For option 10, the PCIC converts the boot address (0000.0000–00FF.FFFF) to the PCI address (f000.0000–F0FF.FFFF) directly in hardware and does not use any of the AFX to PCI translation registers.

- For option 11, the PCIC converts the boot address (0000.0000– 0000.FFFF) to the PCI address (7FFF.0000–7FFF.FFFF) in hardware. Again, this is done without using the AFX to PCI translation registers. The above address conversions are for boot-mode instruction accesses only. Data accesses are treated normally and there are no restrictions while in boot mode.

CHAPTER 12

Error Handling

The microSPARC-IIep CPU must detect and handle many kinds of errors and exceptions. In all CPU error cases, the SPARC IU is interrupted by some type of trap. DMA masters external to the CPU should cause their own IU trap using the PCIC interrupt mechanism. Physical address references to nonexistent addresses in any address space either return indeterminate values or cause timeouts. *Table 12-1* describes the CPU action under the listed error conditions.

Table 12-1 Error Summary

Error	Initiator	Result Summary
Memory parity error	Instruction memory access	set PE, FT=5, L, AT in SFSR cause Instruction Access Error trap (D stage + 1)
	IU, FPU read memory access	set PE, ERR, CP, TYPE in MFSR save PA in MFAR cause L15 interrupt
	IU, FPU write byte, half- word memory access (read-modify-write)	set PE, ERR, CP, TYPE in MFSR save PA in MFAR cause L15 interrupt
Translation error	tablewalk on instruction memory access	set PE, FT=4, L, AT in SFSR cause Instruction Access Error trap (D stage)
Translation error	tablewalk on IU, FPU data memory access	set PE, FT=4, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
Invalid address error	ET=0 during tablewalk on instruction memory access	set FT=1, L, AT in SFSR cause Instruction Access Exception trap (D stage)
	ET=0 during tablewalk on IU, FPU data memory access	set FT=1, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)

Table 12-1 Error Summary *(Continued)*

Error	Initiator	Result Summary
Translation error	ET=3 during tablewalk on instruction memory access	set FT=4, L, AT in SFSR cause Instruction Access Error trap (D stage)
	ET=3 during tablewalk on IU, FPU data memory access	set FT=4, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
Control space error	CPU invalid ASI access	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
	CPU invalid size of access	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
	CPU invalid virtual address during ASI requiring VA	set FT=5, L, FAV, CS in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Privilege violation error (S bit and not ACC 6,7)	IU instruction memory access	set FT=3, L, AT in SFSR cause Instruction Access Exception trap (D stage)
Privilege Violation error (ACC and ASI checked)	IU, FPU data memory access	set FT=3, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Protection error (Memory page ACC and the ASI are checked)	IU, FPU data memory access	set FT=2, L, AT, FAV in SFSR save iu_dva in SFAR cause Data Access Exception trap (R stage)
Protection error (Memory page ACC is checked)	IU, FPU data memory access	set FT=2, L, AT, FAV in SFSR cause Instruction Access Exception trap (D stage)

Note – When a parity error is detected on a DVMA memory read, the level 15 interrupt is set reporting that error. In addition, the MFSR may also incorrectly attempt to report that same error. The information in the MFSR may be invalid in this case and should be cleared and ignored.

APPENDIX **A**

ASI Map

This chapter describes the microSPARC-IIep address space identifier (ASI) map. The ASI is appended to the virtual address by the SPARC IU when it accesses memory. The ASI encodes whether the processor is in supervisor or user mode, and whether an access is to instruction or data memory. The ASI is also used to perform other internal CPU functions.

Table A-1 lists all of the ASI values supported in a microSPARC-IIep system. Only the least significant six bits of the ASI are decoded.

Table A-1 ASI's Supported by microSPARC-IIep

ASI	Function	Acc	Size	Details
00	reserved	-	-	
01-02	unassigned	-	-	
03	Ref MMU Flush / Probe	R / W	single	Section 5.6
04	MMU registers	R / W	single	Section 5.7
05	unassigned	-	-	
06	Ref MMU diagnostics	R / W	single	Section 5.14
07	unassigned	-	-	
08	user instruction	R / W	all	
09	supervisor instruction	R / W	all	
0A	user data	R / W	all	
0B	supervisor data	R / W	all	
0C	instruction cache tag	R / W	single	Section 7.3, Section 7.8
0D	instruction cache data	R / W	single	Section 7.2
0E	data cache tag	R / W	single	Section 6.3, Section 6.10
0F	data cache data	R / W	single	Section 6.2
10	flush I&D cache line (page)	W	single	Section 6.10
11	flush I&D cache line (segment)	W	single	Section 6.10
12	flush I&D cache line (register)	W	single	Section 6.10
13	flush I&D cache line (context)	W	single	Section 6.10
14	flush I&D cache line (user)	W	single	Section 6.10
15–16	reserved	-	-	
17–1C	unassigned	-	-	
1D–1E	reserved	-	-	
1F	unassigned	-	-	
21-3F	reserved	-	-	
40-FF	reserved	-	-	

ASI Descriptions are given in the remainder of this section.

- ASI=0x00

  Reserved — This space is architecturally reserved.

- ASI=0x01-0x02

Unassigned — This space is unassigned and may be used in the future.

- ASI=0x03

  Reference MMU Flush / Probe — This space is used for a flush or probe operation. The virtual address is decoded as follows.

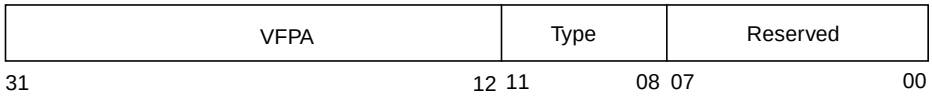


Figure A-1 TLB Flush or Probe Address Format

Field Definitions for ASI 0x03 are given below.

Virtual Flush or Probe Address (VFPA) — This field is the address that is used to index into the TLB. Not all 20 bits are significant, depending on the type of flush or probe.

Type — This field specifies the extent of the flush or the level of the entry probed.

Reserved — These bits are ignored and should be set to zero.

A flush is caused by a single STA instruction and a probe by a single LDA instruction. Flushes are used to maintain TLB consistency by conditionally removing one or more page descriptors. These conditions vary as shown in *Table A-2*.

Table A-2 CPU TLB Entry Flushing

Type	Flush	PTE Match Criteria
0	page	((ACC ≥ 6) OR CID match) AND VA[31:12] match
1	segment	((ACC ≥ 6) OR CID match) AND VA[31:18] match
2	region	((ACC ≥ 6) OR CID match) AND VA[31:24] match
3	context	(ACC ≤ 5) AND CID match
4	entire	None (Entire TLB Flush)
5 to F	reserved	-

Probes cause the MMU to perform a table walk. The table walk stops when a PTE is reached. See *Table 5-11* on page 72.

- ASI=0x04

  Reference MMU Registers — This space is used to read and write internal MMU registers referencing them with a virtual address. Only single word accesses should be used; others cause errors.

Table A-3 Address Map for MMU Registers

VA[12:08]	Register
00	Control register
01	Context Table Pointer register
02	Context register
03	Synchronous Fault Status register
04	Synchronous Fault Address register
05–0F	Reserved
10	TLB Replacement Control register
11–12	Reserved
13	Synchronous Fault Status register[1]
14	Synchronous Fault Address register [1]
15–1F	Reserved [1]

1. Writable for diagnostic purposes

VA bits [31:13] are zero. VA bits [07:00] are ignored and should be set to zero by software.

- ASI=0x05

  Unassigned — This space is unassigned and may be used in the future.

- ASI=0x06

  Reference MMU Diagnostics — Diagnostic reads and writes can be made to the 32 TLB entries using the virtual address to specify which entry and whether the PTE or Tag section is to be referenced.

- ASI=0x07

  Unassigned — This space is unassigned and may be used in the future.

- ASI=0x08

  User Instruction — This space is defined and reserved by SPARC for user instructions.

- ASI=0x09

  Supervisor Instruction — This space is defined and reserved by SPARC for supervisor instructions.

- ASI=0x0A

  User Data — This space is defined and reserved by SPARC for user data.

- ASI=0x0B

Supervisor Data — This space is defined and reserved by SPARC for supervisor data.

- ASI=0x0C

  Instruction Cache Tag — Instruction cache tags are read and written in this space using the LDA and STA instructions at virtual addresses between 0x0 and 0x03FFF on modulo-32 boundaries.

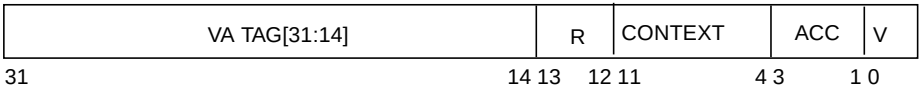


Figure A-2 Instruction Cache Tag Entry

- ASI=0x0D

  Instruction Cache Data — Instruction cache data is read and written in this space using the LDA and STA instructions at virtual addresses between 0x0 and 0x03FFF.

- ASI=0x0E

  Data Cache Tag — Data cache tags are read and written in this space using the LDA and STA instructions at virtual addresses between 0x0 and 0x01FFF on modulo-16 boundaries.

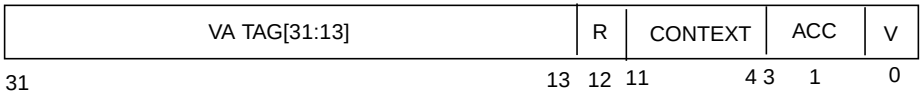


Figure A-3 Data Cache Tag Entry

- ASI=0x0F

  Data Cache Data — Data cache data is read and written in this space using the LDA and STA instructions in ASI 0xF at virtual addresses between 0x0 and 0x01FFF.

- ASI=0x10–0x14

  Flush I & D Cache Line — single cache lines are flushed by directing the STA instruction to one of these spaces.This action removes a single line from both I and D caches.

A cache line is flushed if it meets the minimum criteria given in *Table A-4*.

Table A-4 Flush Criteria for ASI 0x10–0x14

ASI[2:0]	Flush Type	Compare Criterion
0	Page	(S[1] or CNTXT) and VA[31:12]
1	Segment	(S or CNTXT) and VA[13:18]
2	Region	(S or CNTXT) and VA[31:24]
3	Context	U and CNTXT
4	User	U
5,6	reserved	–

1. “S” is the supervisor bit, “U” is the inverse of S, “CNTXT” is the matching of the context register and Tag context, and VA[31:xx] is a comparison based on the virtual address tag.

- ASI=0x15–0x16

  Reserved — This space is architecturally reserved.

- ASI=0x17–0x1C

  Unassigned — This space is unassigned and may be used in the future.

- ASI=0x1D–0x1E

  Reserved — This space is architecturally reserved.

- ASI=0x1F

  Unassigned — This space is unassigned and may be used in the future.

- ASI=0x20

  Reference MMU Bypass —This space can be used to access an arbitrary physical address. It is particularly useful before the MMU or main memory has been initialized. Rather than using the MMU for address translation, a physical address is formed from the least significant 31 bits of the virtual address (PA[30:00] := VA[30:00]). Accesses in bypass mode are not cacheable.

- ASI=0x21–0x2F

  Reserved — This space is architecturally reserved.

- ASI=0x30–0x38

  Unassigned — This space is unassigned and may be used in the future.

- ASI=0x39

  Data Cache Diag Register — This space is used to read and write internal data cache registers using the virtual address to reference them. Single word accesses only should be used, others result in an error.

- ASI=0x3A–0x3F

Unassigned — This space is unassigned and may be used in the future.

- ASI=0x40–0xFF

  Reserved — Since the two high order bits are not decoded this range of encodings should not be used. If it is used, only the lower six bits are decoded; the two upper bits are ignored.

APPENDIX **B**

Physical Memory Address Map

The physical address space for microSPARC-IIep is mapped into eight address spaces based on the upper three bits of the physical address(PA[30:28]). *Table B-1* shows the address space mapping against the values of these bits.

Table B-1 Physical Address Space

PA[30:28]	Address Space
000	main memory space (256 MB)
001	control space (Sun-4m system registers, 256 MB)
010	flash memory space (256 MB)
011	PCI space (256 MB)
100	reserved I/O space: should not be accessed
101	reserved I/O space: should not be accessed
110	reserved I/O space: should not be accessed
111	reserved I/O space: should not be accessed

APPENDIX **C**

microSPARC-IIep AFX (Local) Bus

C.1 Introduction

The AFX (Local) bus employs a memory-level interconnect protocol between the microSPARC-IIep CPU, system memory, and subsystems. This local bus provides high bandwidth, low latency, and slave-only access by placing the interface directly on the system memory bus in a uniprocessor environment. An internal-only version of the local bus (not available to the external interface) is used to supply the interface to the PCIC. A device on this internal version of the bus can request its use as a bus master. This section describes some of the operations of the local bus.

The principal Local Bus features are:

- 64-bit data path
- 28-bit physical address per slave
- Clock rate of up to 75 MHz
- Synchronous operation, except for occurrence of interrupts
- One, two, four, or eight-byte transfers
- Single interrupt line per slave

The local bus provides high-speed access between the processor, system memory, and devices. All accesses to the system memory and Local Bus slaves are controlled by the microSPARC-IIep processor. The local bus controller, simply referred to as the controller in this document, is integrated within the microSPARC-IIep chip. Any local bus slave can request an interrupt of this controller, but only the controller can perform read or write accesses. Figure A.1 shows a block diagram of the local bus.

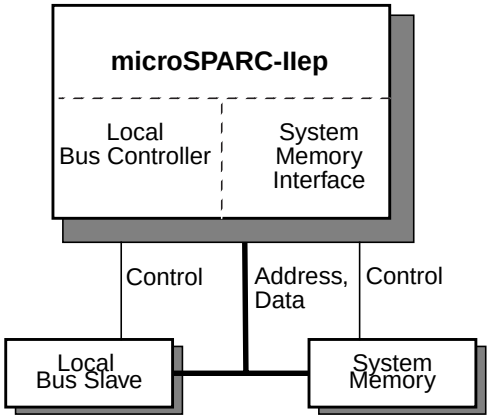


Figure C-1 Local Bus Block Diagram

C.1.1 System Memory Interface

Since the local bus shares the system memory address and data lines, the local bus controller and system memory interface must use arbitration for allocation of these common lines. The memory interface is integrated within the microSPARC-IIep chip.

C.1.2 Local Bus Controller

This controller is the single master of the local bus and is responsible for initiating each local bus cycle. The local bus controller is integrated within the microSPARC-IIep chip.

C.1.3 Local Bus Slave

The local bus slave responds to requests given by the controller. The local bus slave can interrupt the controller using a single asynchronous interrupt line.

The local bus may have multiple slave devices. Each local bus slave is independent of any other local bus slave.

C.1.4 Local Bus Interface

Table C-1 shows the interface signals that connect the microSPARC-IIep chip to a local bus slave. These signals are either generated by the microSPARC-IIep controller or by the local bus slave. DB[63:0] bus signal lines may be driven by either the microSPARC-II CPU or by the slave, depending on the type of bus cycle.

Table C-1 Local Bus Interface Signals

Signal Name	Description	Driven By
CLK	clock	controller
AEN	address enable	controller
LO_ADDR	low address select	controller
WRITE_L	read / write select	controller
AB[14:0]	address / byte mask bus	controller
P_REPLY[1:0]	port reply	slave
S_REPLY[1:0]	system reply	controller
DB[63:0]	data bus	controller / slave
RESET_L	reset	controller

C.2 Basic Local Bus Cycle

A local bus slave has a minimum four-deep request FIFO that holds address, size, and—for write requests—data. There may be at most one outstanding read request in the FIFO at any time. Write requests sent to the slave should be saved in the FIFO and later acknowledged. Read requests must allow the FIFO to drain before responding.

The local bus controller must assure a FIFO depth of four slots, but may optionally probe the slave after a power-on reset to determine the exact size of its FIFO. The controller throttles access requests to the slave. There are no FIFO-full or FIFO-empty signals between the controller and slave.

A complete local bus cycle consists of two major phases: address and data. These phases may operate independently of each other. Addresses of successive accesses may be sent to the slave without regard for the associated data, provided the slave's FIFOs are not overloaded. Data transfers are completed using the reply control signals, allowing FIFO entries in the slave to be available again.

C.2.1 Address Cycles

Local bus address cycles send physical address, access size, and access direction in two separate cycles. These cycles are controlled by the address enable signal AEN. The cycle type is denoted by the state of the LO_ADDR signal. See *Figure C-2*.

In address cycle 0, the upper bits of the physical address are placed on the address bus (AB) lines. In address cycle 1, the lower bits of the physical address and the access size are placed on the AB lines, while the access direction (read or write) is defined by the WRITE_L signal.

For subsequent accesses, address cycle 0 need not be repeated if the upper physical address bits do not change. The slave must latch this data and use it until the controller sends the next address cycle 0. This functionality is similar to the *Page Mode* feature of DRAMs.

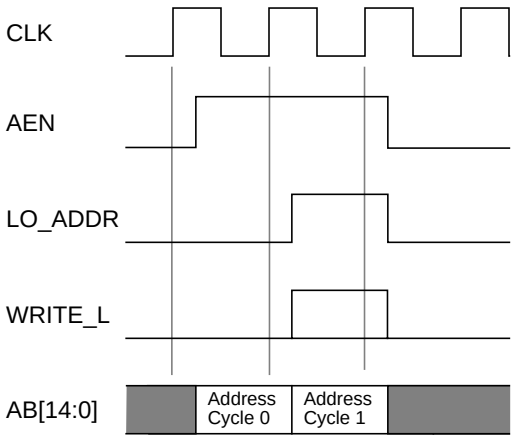


Figure C-2 Address Cycles

C.2.2 Data Cycles

Each time an address cycle 1 is sent, the read or write command to the slave should be considered launched and a data cycle may be initiated.

The local bus cycle must complete within 2048 local bus clock cycles of the launching address cycle. The local bus controller times-out if the slave has not responded with the appropriate P_REPLY (port reply) code.

A slave must always acknowledge accesses to its entire address range using the appropriate P_REPLY code (read single or write single). Since the local bus does not specify an error acknowledge, slaves may provide a readable location that identifies when error accesses have occurred. These error types may include out-of-bounds access, unsupported size-type access, and so on. Slaves may also interrupt the controller to identify error accesses.

C.2.2.1 Write

The controller initiates the write data cycle by placing the *Write Single* code (10) on the S_REPLY (system reply) lines. This may happen in the same clock as the launching address cycle 1, or any clock cycle following. Once sent, the controller places the write data on the DB (data bus) lines in the next clock cycle. The slave must acknowledge the write by placing the Write Single code (10) on the P_REPLY lines in the next, or later, clock cycle.

C.2.2.2 Read

In the *read* cycle, the CPU issues a *Read Single* code (11) on the S_REPLY lines in the same clock cycle as the launching address cycle 1. When the slave is ready to acknowledge the read, the slave drives the data on the DB lines and a Read Single code (11) on the P_REPLY lines in the same clock cycle. Use of this mode gives the advantage of doing a best-case read in two clock cycles.

C.2.3 Local Bus Timeout

The local bus controller supports a timeout mechanism to prevent the system from waiting on a broken or absent local bus slave device. The timeout period for an access is measured from the positive edge of the CLK signal which launches a Local access (address cycle 1). This period is defined as 2048 CLK cycles (this time interval must be greater than 10 microseconds) before the corresponding P_REPLY is returned by the slave.

If a timeout occurs, the controller aborts the current cycle and all outstanding local bus cycles.

C.2.4 Local Bus Latency

For read cycles, an local bus slave should respond to accesses with an average latency of 1.0 microseconds, and a worst case latency of 2.0 microseconds.

To enforce this condition, the controller must hold off write requests until a slot is available in the slave's write FIFO. The controller must also hold off read requests until the slave's write FIFO has zero or one writes pending.

The threshold is settable at the user level and exists as part of the physical address control space on its own page. If the threshold is zero, the latency for a read is set by the slave's maximum read time. If the threshold is one, the maximum latency is for one read and one write retirement. The default value default at power on is zero.

If a local bus slave implements features that would cause a write-read combination to violate the above worst-case latency, it should support a software polling mechanism. Immediate access register(s) should be provided to respond to inquiries about the status of the slave device.

Local bus interrupt latency should nominally be less than 10 microseconds and less than 50 microseconds for a worst case.

C.3 Local Memory Map

Each slave occupies 256 megabytes of address space in a system memory map.

C.4 Local Bus Interconnect

Figure A.3 shows the local bus interconnect and all of the associated signals. In the case where a system has multiple local bus slaves, some signals are required to be unique to each one. These are noted as *radial* (R). For an efficient implementation, other signals may be common between connectors. These are noted as *bused* (B).

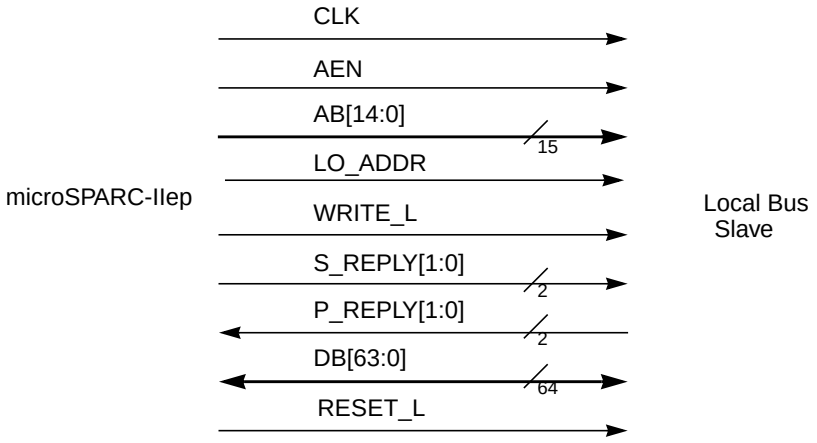


Figure C-3 Local Bus Signals

Table C-2 Local Bus Signal Summary

Bus Signal Name	I/O	Description	Driven By	Multiple Slaves[1]
CLK	I	clock	controller	B[1]
AEN	I	address enable	controller	R [1]
LO_ADDR	I	low address select	controller	B
WRITE_L	I	read / write select	controller	B
AB[14:0]	I	address / byte mask bus	controller	B
P_REPLY[1:0]	O	port reply	slave	R

Table C-2 Local Bus Signal Summary *(Continued)*

Bus Signal Name	I/O	Description	Driven By	Multiple Slaves[1]
S_REPLY[1:0]	I	system reply	controller	R
DB[63:0]	I/O	Data bus	Controller/ Slave	B
RESET_L	I	Reset	Controller	B

1. R = Radial, B = Bused

C.5 Local Bus Signals

This section provides detailed descriptions of the local bus signals.

C.5.1 CLK

Local Bus Clock: this signal is generated by the microSPARC-IIep CPU and its frequency is within the range of 25 MHz to 42 MHz. The clocks may be used in differential form to improve the common mode noise immunity at high clock rates. The maximum clock frequency in the single-ended mode is 42 MHz. Typical system clock rates can exceed 20 MHz.

C.5.2 AEN

Address Enable: when asserted (high), this signal indicates that there is a command request and there is valid data on the AB bus and the LO_ADDR and WRITE_L lines. The AEN signal also qualifies read cycles.

C.5.3 LO_ADDR

Low Address: this signal is qualified by the AEN (address enable) signal. LO_ADDR defines which address cycle the controller is sending on the address bus and write lines. When driven to 0, this signal indicates address cycle 0. When driven to 1, it indicates address cycle 1 (low address bits and byte mask). See *Table C-3*.

Table C-3 LO_ADDR Signal States

LO_ADDR State	Address Cycle	Comments
0	0	high address bits
1	1	low address bits and byte mask

C.5.4 WRITE_L

Write: this signal is qualified by AEN (address enable) and is only valid during address cycle 1. When driven to 0, this signal indicates a write request. When driven to 1, this signal indicates a read request.

C.5.5 AB[14:0]

Address/Byte Mask bus: This bus conveys the multiplexed physical address of where the data is to be written to or read from together with the byte mask. This data is multiplexed over two address cycles, as shown in *Table C-4*.

C.5.6 Byte Mask (BM) Bits

Table C-4 Address Bus Multiplexing

Address Bus	Cycle 0	Cycle 1[1]
LO_ADDR	0	1
WRITE_L	X	R / W
AB[14]	PA[27]	BM[3]
AB[13]	PA[26]	BM[2]
AB[12]	PA[25]	BM[1]
AB[11]	PA[24]	BM[0]
AB[10]	PA[23]	PA[13]
AB[9]	PA[22]	PA[12]
AB[8]	PA[21]	PA[11]
AB[7]	PA[20]	PA[10]
AB[6]	PA[19]	PA[9]
AB[5]	PA[18]	PA[8]
AB[4]	PA[17]	PA[7]
AB[3]	PA[16]	PA[6]
AB[2]	PA[15]	PA[5]
AB[1]	PA[14]	PA[4]
AB[0]	X	PA[3]

1. PA = Physical address; BM[3:0] = Byte mask bits; X = don't care; R / W = Read or write

The byte mask bits, BM[3:0], indicate the transfer size, as shown in *Table C-5*,

Table C-5 Byte Mask (BM) Bits

Mask Bits	Data								
BM[3:0][1]	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]	Comment
0000	R / W	--	--	--	--	--	--	--	Byte access 0
0001	--	R / W	--	--	--	--	--	--	Byte access 1
0010	--	--	R / W	--	--	--	--	--	Byte access 2
0011	--	--	--	R / W	--	--	--	--	Byte access 3
0100	--	--	--	--	R / W	--	--	--	Byte access 4

Table C-5 Byte Mask (BM) Bits

Mask Bits	Data								
BM[3:0][1]	[63:56]	[55:48]	[47:40]	[39:32]	[31:24]	[23:16]	[15:8]	[7:0]	Comment
0101	--	--	--	--	--	R/W	--	--	Byte access 5
0110	--	--	--	--	--	--	R/W	--	Byte access 6
0111	--	--	--	--	--	--	--	R/W	Byte access 7
1000	R/W	R/W	--	--	--	--	--	--	Half word 0
1010	--	--	R/W	R/W	--	--	--	--	Half word 1
1100	--	--	--	--	R/W	R/W	--	--	Half word 2
1110	--	--	--	--	--	--	R/W	R/W	Half word 3
1001	R/W	R/W	R/W	R/W	--	--	--	--	Word 0
1101	--	--	--	--	R/W	R/W	R/W	R/W	Word 1
1011	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	Double word
1111	--	--	--	--	--	--	--	--	Reserved

1. The BM codes in the left column are not in absolute numerical order.

C.5.7 Multiplexed Addresses

After any reset issued to the slave, the controller establishes a full physical address for the first access request to the slave. Since the physical address is multiplexed across two address cycles, both address cycle 0 and address cycle 1 must be issued to the slave.

The slave must always latch the upper physical address bits from address cycle 0 to combine them with the lower physical address bits in each following address cycle 1. Another address cycle 0 only needs to be issued when the data access crosses the page boundary and the upper address bits need to be modified.

The local bus controller is allowed to issue extraneous address cycle 0s, but it is recommended that their generation be minimized for better system performance.

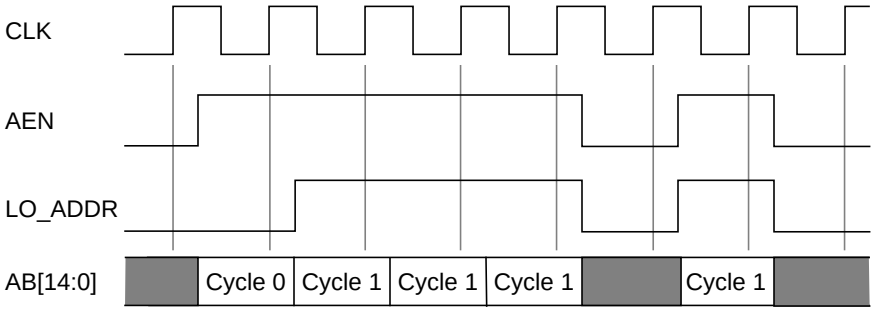


Figure C-4 Multiplexed Addresses

C.5.8 P_REPLY[1:0]

Port reply: These output signals indicate that the data has been processed, that is, removed, from the write buffer on writes. For reads they indicate that the read data is available in the read latch.

Table C-6 P_REPLY[1:0] Signals

P_REPLY[1]	P_REPLY[0]	Type of Access
0	0	Idle
0	1	Reserved
1	0	Write single (CE) P_WAS
1	1	Read single (OE) P_RAS

C.5.9 S_REPLY[1:0]

System reply: these signals indicate that the local bus has been selected and they show the type of access that is being made. See *Table C-7*.

Table C-7 S_REPLY[1:0] Signals

S_REPLY[1]	S_REPLY[0]	Type of Access
0	0	Idle

Table C-7 S_REPLY[1:0] Signals

S_REPLY[1]	S_REPLY[0]	Type of Access
0	1	Idle
1	0	Write single (CE) S_WRS
1	1	Read single (OE) S_SRS

On writes, the data appears in the following cycle. In read cycles, the data is enabled onto the bus in the same cycle as the P_REPLY signal is driven. See *Figure C-5*.

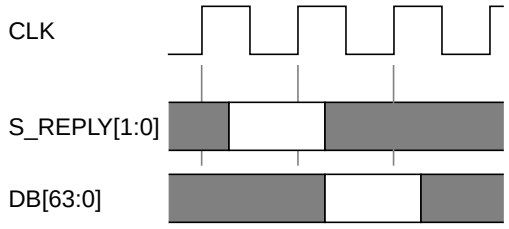


Figure C-5 S_REPLY[1:0] Signal

C.5.10 DB[63:0]

Data bus: This bus carries the data being transferred between the microSPARc-IIep CPU and the local bus slave. Every local bus must carry all 64 data-bus signals, over which data is transferred. The local bus supports four primary data formats:

- Bytes, which consist of eight data bits
- Half words, which consist of 16 data bits
- Words, which consist of 32 data bits
- Double words, which consist of 64 data bits

By convention, the least-significant bit of the data bus is DB[0] while the most-significant bit is DB[63]. These pins have internal pullups.

The local bus uses *big-endian* addressing. As shown in *Figure C-6*, big-endian addressing means that the significance of bytes in a half-word, word, or double-word decreases as the address increases. The byte ordering is specified by the byte mask bits in the address.

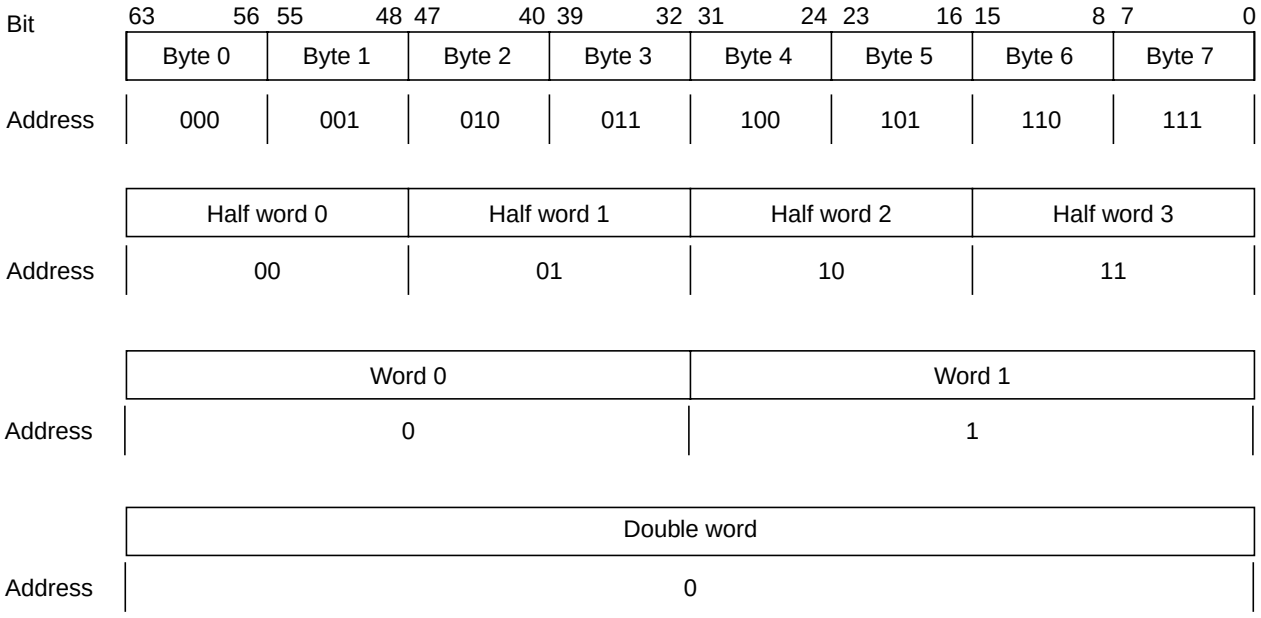


Figure C-6 Data Bus Byte Ordering

C.5.11 RESET_L

Local Bus Reset: the $\overline{\text{RESET}}$_L signal properly initializes all local slaves after power-up and system reset. In all cases, RESET_L must be asserted for at least 512 clock cycles before being unasserted. In the case of power-on, RESET_L must be stable before these 512 clock cycles begin. The leading edge of RESET_L may or may not meet setup times with respect to CLK. The trailing edge of RESET_L must meet setup and hold times with respect to CLK. The local bus controller may keep RESET_L asserted for more than 512 clock cycles, if required.

Upon detecting the assertion of RESET_L, a local bus slave must perform whatever internal operations are required to initialize itself. While RESET_L is asserted, a local bus slave must not drive bused signals and must drive radial signals to an inactive state.

C.6 Local Bus Timing Diagrams

This section shows the timing diagrams for write and read cycles on the local bus for the microSPARC-IIep CPU.

C.6.1 Write Cycle

In a write, the controller places the address on AB[14:0] and asserts AEN. This example shows a two-cycle address, so the controller asserts LO_ADDR for address cycle 1 (the low address). The controller asserts S_REPLY either simultaneous with the second address or up to n cycles later.

The controller places the write data on DB[63:0] on the next cycle after receiving the S_REPLY write single (10) code. The slave acknowledges receipt of the data by asserting the P_REPLY write single (10) code.

Figure C-7 shows the timing for a fast write. *Figure C-8* shows the timing for a slow write.

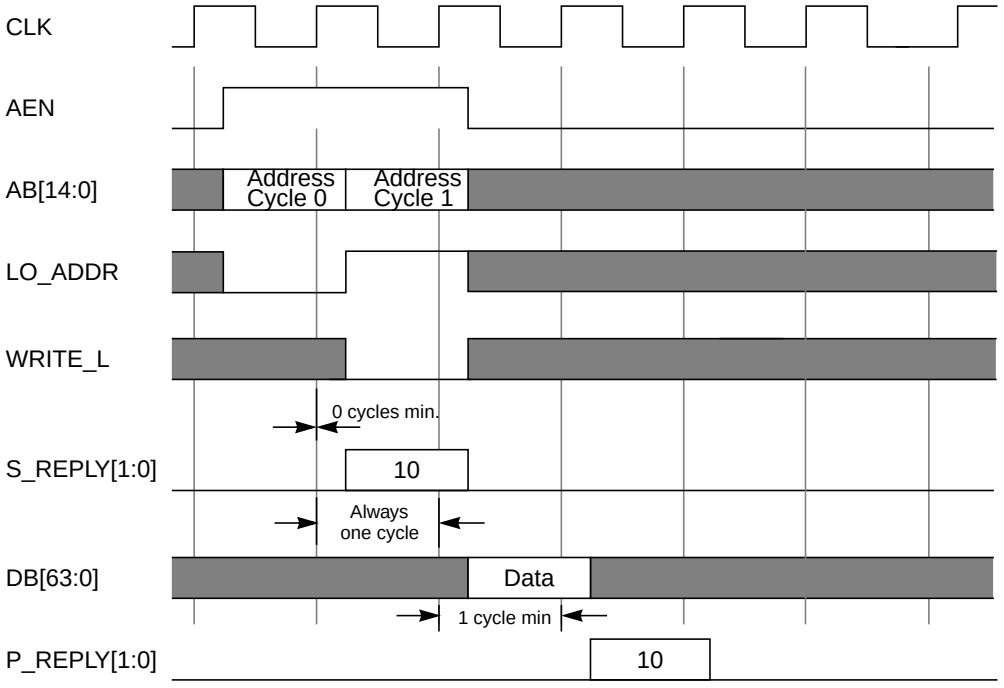


Figure C-7 Fast Write Timing

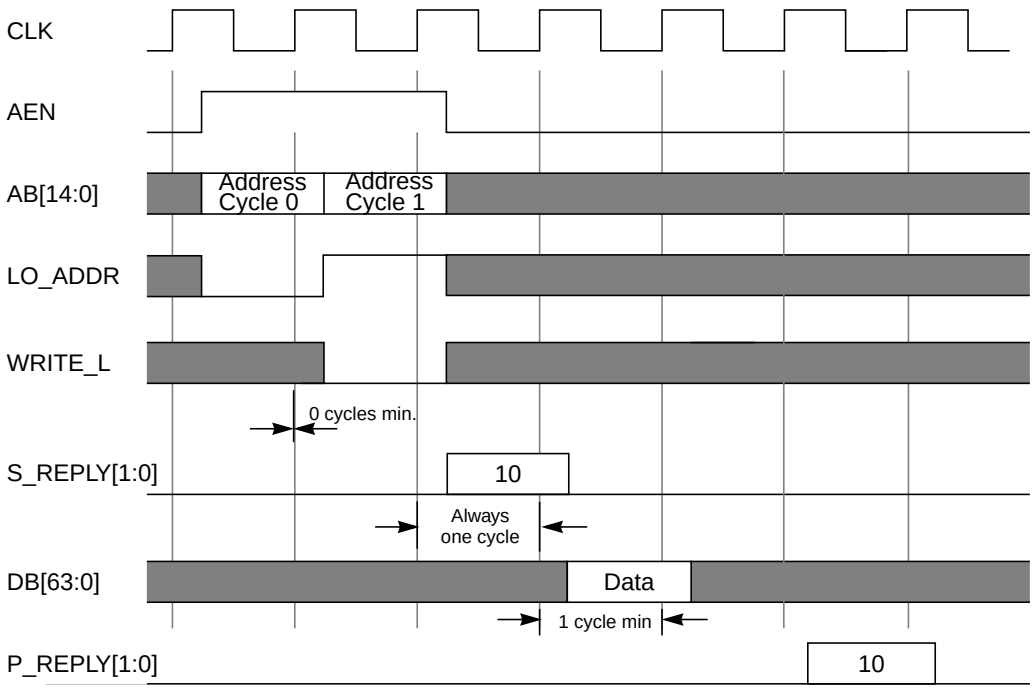


Figure C-8 Slow Write Timing

C.6.2 Read Cycle

Figure C-9 shows the read timing. In the read, the controller places the address on AB[14:0] and asserts AEN. On the second address cycle (address cycle 1), the controller deasserts WRITE_L and asserts LO_ADDR and the S_REPLY read single (11) code.

As soon as it has the data available, the slave responds to the read request by placing the requested read data on DB[63:0] and asserting the P_REPLY read single (11) code.

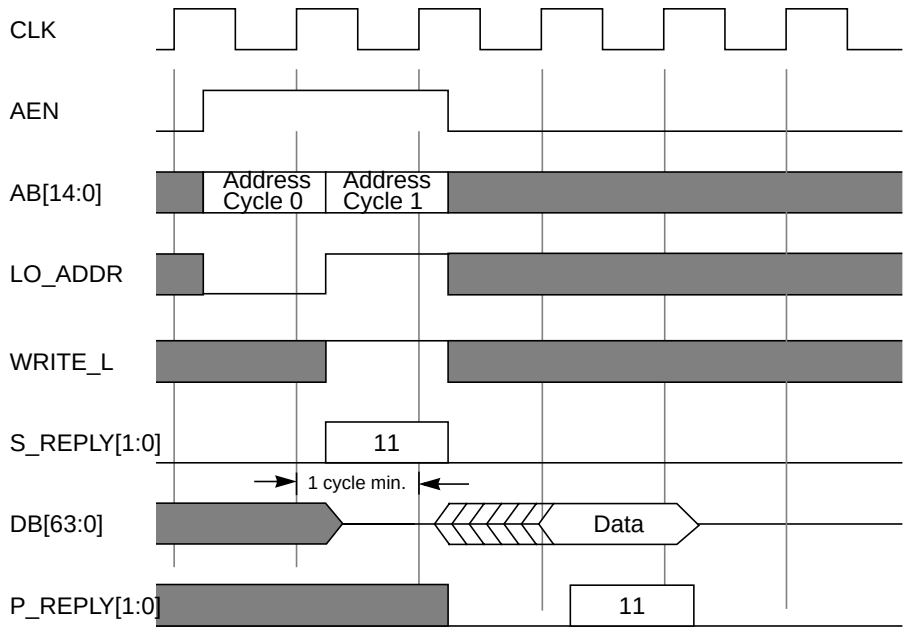


Figure C-9 Read Cycle Timing

C.7 Back-To-Back Write and Read Cycles

Figure C-10 shows timing for back-to-back writes and reads. This example shows the fastest possible controller and slave cycles. It also assumes the read launching bit in the CPU is set to one.

The letter “bubbles”, for example, A, B, and C, denote the pipelining of the command. A through D are write cycles. E and F are read cycles.

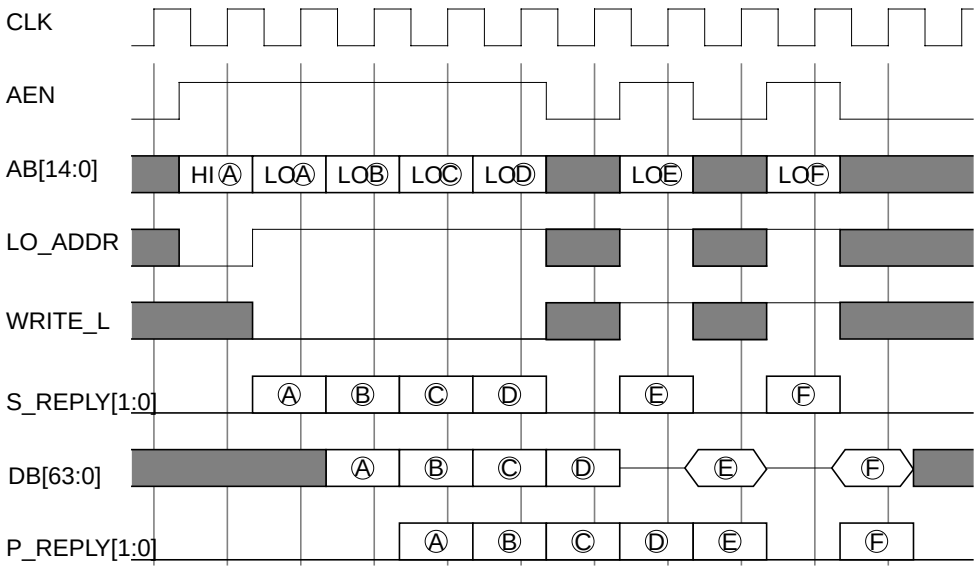


Figure C-10 Back-To-Back Write and Read Timing

APPENDIX **D**

Memory Timing Parameters

D.1 Tabulated Parameter Values

These parameter values, for a variety of primary processor clock frequencies, are given in *Table D-1* on page 274.

Table D-1 microSPARC-IIep Memory Timing Parameters

Parameter	Spec (ns)	70Mhz (ns)	85Mhz (ns)	100Mhz (ns)	125Mhz (ns)	150Mhz (ns)	175Mhz (ns)	200Mhz (ns)
t_RP	40	3.5 (50.0))	3.5 (41.2)	4.5 (45.0)	5.5 (44.0)	6.5 (43.3)	7.5 (42.8)	8.5 (42.5)
t_RAS (rd)	60	7.5 (107.2)	8.5 (100.0)	9.5 (95.0)	11.5 (92.0)	13.5 (89.9)	16.5 (94.2)	18.5 (92.5)
t_RAS (wr)	60	5.5 (78.6)	8.5 (100.0)	8.5 (85.0)	9.5 (76.0)	12.5 (83.2)	14.5 (82.8)	15.5 (82.5)
t_CP (rd)	10	1 (14.3)	1 (11.8)	2 (20.0)	2 (16.0)	2 (13.3)	3 (17.1)	3 (15.0)
t_CP (wr)	10	2 (28.6)	3 (35.3)	3 (30.0)	3 (24.0)	4 (26.6)	4 (22.8)	4 (20.0)
t_CAS (rd)	15	3 (42.9)	4 (47.1)	4 (40.0)	5 (40.0)	6 (40.0)	7 (40.0)	8 (40.0)
t_CAS (wr)	15	2 (28.6)	3 (35.3)	3 (30.0)	3 (24.0)	4 (26.6)	5 (28.5)	5 (25.0)
t_ASC	4	1 (14.3)	3 (35.3)	3 (30.0)	4 (32.0)	4 (26.6)	6 (34.3)	7 (35.0)
t_RAD, RAH	15-25, 10	1.5 (21.5)	1.5 (17.6)	1.5 (15.0)	1.5 (12.0)	2.5 (16.6)	2.5 (14.3)	2.5 (12.5)
t_RCD (rd)	20-40	3.5 (50.1)	3.5 (41.1)	4.5 (45.0)	5.5 (44.0)	6.5 (43.3)	8.5 (48.5)	9.5 (47.5)
t_RCD (wr)	20-40	2.5 (35.8)	4.5 (52.9)	4.5 (45.0)	5.5 (44.0)	7.5 (49.9)	8.5 (48.5)	9.5 (47.5)
t_DS, WCS	0, 4	1 (14.3)	3 (35.3)	3 (30.0)	4 (32.0)	4 (26.6)	6 (34.3)	7 (35.0)
t_DH,WCH	20, 19	2 (28.6)	3 (35.3)	3 (30.0)	3 (24.0)	4 (26.6)	5 (28.5)	5 (25.0)
t_RPC (ref)	10	2 (28.6)	2 (23.6)	2 (20.0)	2 (16.0)	3 (20.00)	3 (17.1)	3 (15.0)
t_CSR (ref)	15	1.5 (21.5)	1.5 (17.6)	2.5 (25.0)	3.5 (28.0)	3.5 (23.3)	4.5 (25.6)	5.5 (27.5)
t_CHR (ref)	20	4.5 (54.4)	4.5 (52.9)	4.5 (45.0)	6.5 (52.0)	8.5 (56.6)	9.5 (54.1)	10.5 (52.5)
t_RAS (ref)	60	6.5 (92.9)	6.5 (70.7)	6.5 (65.0)	8.5 (68.0)	10.5 (89.9)	11.5 (65.6)	12.5 (62.5)
t_RAS (rmw)	111	13.5 (193.0)	15.5 (182.9)	17.5 (175.0)	18.5 (148.0)	24.5 (163.2)	28.5 (162.7)	32.5 (162.5)
t_CAS1 (rd) (rmw)		3 (42.9)	4 (47.1)	4 (40.0)	5 (40.0)	6 (40.0)	7 (40.0)	8 (40.0)
t_CAS2 (wr) (rmw)		2 (28.6)	3 (35.3)	3 (30.0)	3 (24.0)	3 (20.0)	5 (28.5)	5 (25.0)
t_CP (rmw)		4 (57.2)	5 (58.9)	6 (60.0)	7 (56.0)	9 (59.8)	10 (57.1)	11 (55.0)
sp_sel		000	001	010	011	100	110	111

Bibliography

General References

Books and Specifications

[Weaver, David L., editor.] *The SPARC Architecture Manual, Version 8*, Prentice-Hall, Inc., 1992.

IEEE Standard Test Access Port and Boundary-Scan Architecture, IEEE Std 1149.1-1990, IEEE, New York, NY, 1990.

PCI Local Bus Specification, Revision 2.1: PCI Special Interest Group, Portland OR, June 1995.

Sun Microelectronics (SME) Publications

These books and papers are available in printed form, and some are also available through the World Wide Web. See “On Line Resources” below for information about the SME WWW pages.

Data Sheets

microSPARC-IIep Highly Integrated 32-bit RISC/PCI Microprocessor Data Sheet, 802-7327-03, February, 1997

User's Guides

microSPARC Microprocessor User's Manual, STP1010TAB50, Rev. 1.0, June 1994

microSPARC-II Microprocessor User's Guide, STP1012PGA-UG, Rev 1.1, July 1994

microSPARC-II Microprocessor User's Guide Errata, STP1012-UGE

microSPARC-IIep Megacell Reference, Part No: 806-1955-01 (note that this is part of the CSL release.)

How to Contact Sun Microelectronics

Sun Microelectronics can be reached at:

Sun Microsystems, Inc.
Microelectronics
901 San Antonio Road
Palo Alto, CA, U.S.A. 94303
Tel: 800 681-8845

On Line Resources

The Sun Microelectronics WWW page is located at:

http://www.sun.com/microelectronics

It contains the latest information about the entire SPARC product line, including the microSPARC-IIep CPU.

Index

E

F

I

J

L

M

N

O

P

R

S

T

U

V

W

Y

Sun Microsystems
Microelectronics
901 San Antonio Road
Palo Alto, CA 94303–4900 USA
800 / 681-8845
www.sun.com / microelectronics

Part Number: 802-7100-02