TMS390S10

microSPARC™

Reference Guide

Highly Integrated SPARC for Low-Cost Desktop Solutions

TEXAS INSTRUMENTS

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to or to discontinue any semiconductor product or service identified in this publication without notice. TI advises its customers to obtain the latest version of the relevant information to verity, before placing orders, that the information being relied upon is current.

TI warrants performance of its semiconductor products to current specifications in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Unless mandated by government requirements, specific testing of all parameters of each device is not necessarily performed.

TI assumes no liability for TI applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

TI products are not intended for use in life-support appliances, devices or systems. Use of a TI product in such applications without the written consent of the appropriate TI officer is prohibited.

Table of Contents

List of Figures

List of Tables

Preface

This guide contains application information for the highly integrated SPARC processor, named the TMS390S10. Throughout this guide the term microSPARC is used to describe the TMS390S10 chip.

This User's Guide should be used in conjunction with the TMS390S10 data sheet. Where conflicts between these documents exist, particularly in reference to exact timings and frequency information, the TMA390S10 data sheet has precedence.

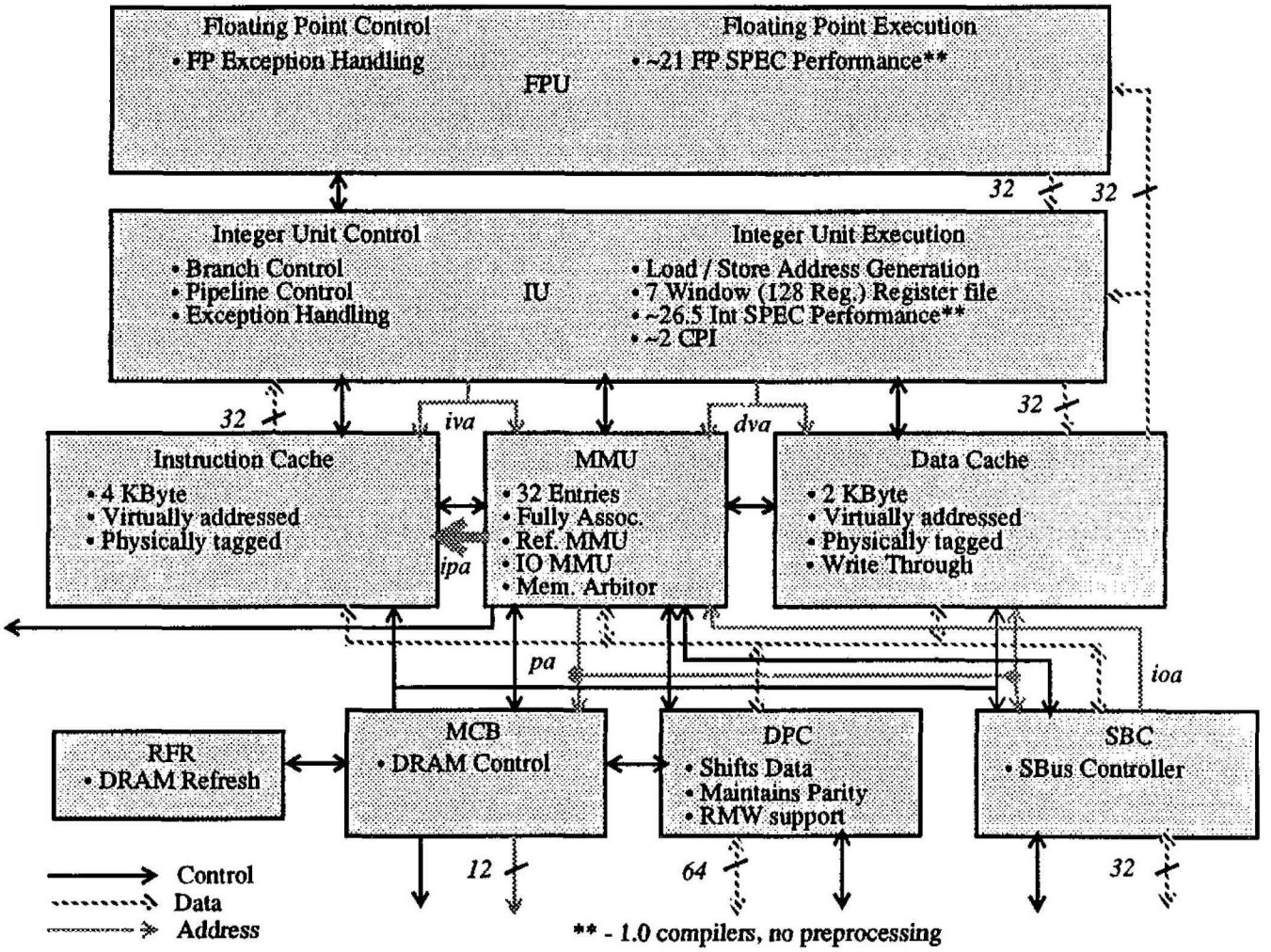
1.0 Overview

The microSPARC CPU is a highly integrated, low cost implementation of the SPARC RISC architecture. High performance is achieved by the high level of integration including on chip instruction and data caches and the close coupling of the CPU with main memory. A full custom implementation allows for a target frequency of 50MHz providing sustained performance of 24 Specmarks with 1.0 rev. compilers and no preprocessing. The design is highly testable with the use of the full JTAG scan support. The microSPARC chip will support up to 128MB of DRAM and 4 SBus slots.

Integrated within microSPARC are a SPARC V8 Integer Unit core, a SPARC Reference Memory Management Unit, a Floating Point Unit, Instruction and Data Caches, DRAM controller, and an SBus Controller

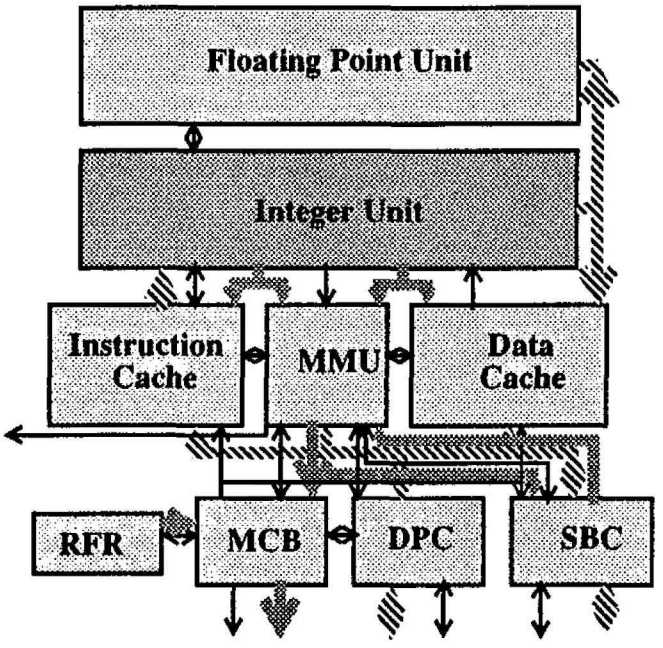
A simple block diagram follows.

Figure 1.0 - microSPARC Block Diagram



2.0 Integer Unit

The microSPARC integer unit is a SPARC integer unit as defined in the SPARC architecture manual (version 8). The IU design goal is to maximize performance, given a constrained die size, using a predefined software architecture. The emphasis is on software compatibility, since the greatest cost impact would be on any software (i.e. kernel, compilers) that would need rewriting.

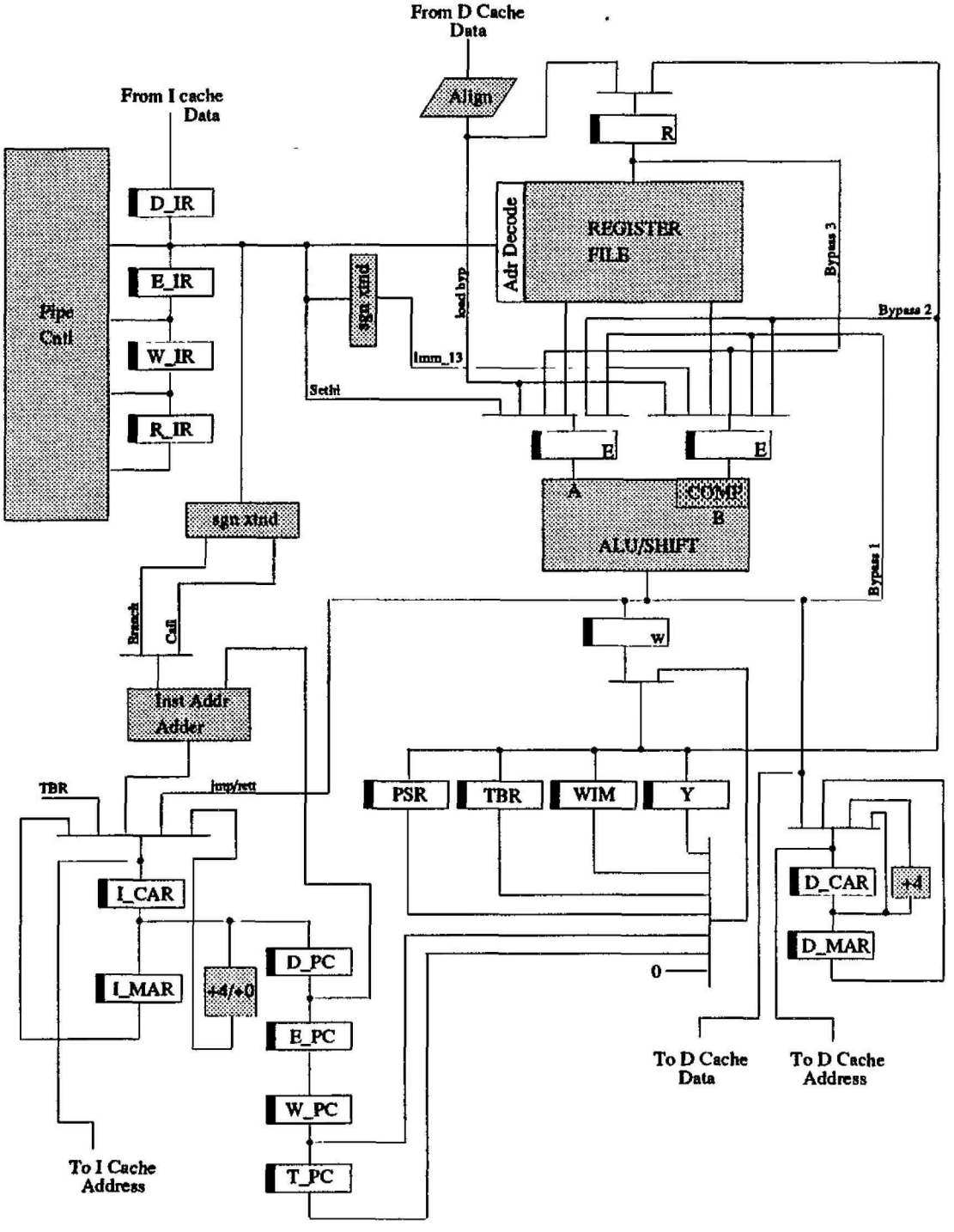


2.0.1 Overview

The microSPARC integer unit is a CMOS implementation of the SPARC 32-bit (Rev 8) RISC architecture. Some important features of this design are:

- Single issue, 5 stage pipeline
- Harvard architecture
- Instruction and Data cache streaming support
- IMUL and IDIV implemented as integer operations
- 0 cycle branch penalty
- 120-register register file (7 register windows)

Figure 2.0 - microSPARC IU Block Diagram



2.0.2 Instruction Pipeline

The microSPARC IU uses a single instruction issue pipeline with 5 stages.

F (Instruction Fetch): Instruction cache access occurs in this cycle based on the address generated in the previous cycle. The instruction is valid on the pins of the IU at the end of this cycle and are registered inside of the IU.

D (Decode): The decode stage is used to decode the instruction and to read the necessary operands. Operands may come from the register file or from internal data bypasses. The register file has 2 independent read ports For situations where the necessary operand is in the pipeline and has not yet been written to the register file, internal bypasses are supplied to prevent pipeline interlocks. In addition, addresses are computed for CALL and Branch in this cycle in the address adder.

E (Execute): The execute stage is used to perform ALU, logical, and shift operations. For memory operations (e g.: LD) and for JMPL/RETT the address is computed in this cycle.

W (Write): This stage is used to access the data cache. For cache reads, the data will be valid by the end of this cycle, at which point it is aligned as appropriate. For cache writes, the data is presented to the data cache in this cycle.

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

Table 2.0 - Cycles per Instruction

Instruction	Cycles	Words
Call	1	1
Single Loads	1	1
Jump/Rett	2	1
Double Loads	2	1
Single Stores	2	1
Double Stores	3	1
Taken Trap	3	1
Atomic Load/Store	2	1
SWAP	2	1
All Others	1	1

2.0.3 Memory Operations

2.0.3.1 Loads

All load operations take 1 cycle in the microSPARC IU except for LDD which takes 2. For LD, LDB, and LDH the pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E - Address operands are added to yield the memory address This address is passed to the cache in this cycle.

W - Address is registered in the cache and access is started. Data is expected at the end of this cycle. Any necessary alignment and sign extension is done in the IU prior to being registered.

R - Data is registered in the IU and is written into the register file.

In the event of a cache miss, the miss indication is given to the IU in the R cycle. It is flagged early enough to prevent writing bad data to the register file. The pipe is held and the miss address is resent to the cache to service the miss. The cache indicates when the miss data is available - the IU can then register it into the appropriate R cycle register and write it into the register file.

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

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E - Address operands are added to yield the even memory address This address is passed to the data cache in this cycle.

W (E2) - 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.

R (W2) - 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. This data gets sent to the IU

R2 - Odd word is registered in the IU and written to the register file.

In the event of a cache miss, the miss indication is in the R cycle of the LDD (the same as the W cycle of the LDD's help cycle). The miss is indicated early enough to prevent writing bad data into the even register. The pipe is held and the even address is resent to the cache. When the cache sends the correct data, the R register is written with the correct data and the odd address is resent to get the odd word.

2.0.3.2 Stores

The microSPARC IU register file has only two independent read ports As a result, store operations take 2 cycles, except STD which takes 3. For ST, STB, and STH the pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the memory address. This address will be registered within the IU to provide the data cache with the address in the correct cycle. At the same time, the store data is read from the register file or bypassed from instructions still in the pipe.

W (E2) - The store address is sent to the data cache.

R (W2) - The store data is sent to the data cache in this cycle along with the appropriate byte marks.

R2 - Store is complete.

For STD the pipeline does the following:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the even memory address. This address will be registered within the IU to provide the data cache with the address in the correct cycle. At the same time, the even store data is read from the register file or bypassed from instructions still in the pipe.

W (E2/D3) - Even address is sent to the data cache. Odd word is read from register file.

R (W2/E3) - Even store data is sent to the data cache. Odd address is sent to the data cache.

R2 (W3) - Odd data is sent to the data cache.

R3 - STD complete.

2.0.3.3 Atomics

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

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

E (D2) - The address operands are added to compute the swap address. This address is sent to the data cache to start the cache read portion of the operation. The address is also registered inside of the IU to provide the data cache with the same address

for the store in the next cycle. The register to be swapped is read out in this cycle.

W (E2) - The data cache returns the memory location accessed. The store address is sent to the data cache.

R (W2) - The IU registers the read data and writes it to the register file. Also the store data is sent to the data cache.

R2 - SWAP complete.

The pipeline does the following on the LDSTUB instruction:

D - Register operands are read from the register file or are bypassed from instructions still in the pipe. An immediate operand is sign extended.

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. The address is also registered inside of the IU to provide the data cache with the same address for the store in the next cycle.

W (E2) - The data cache returns the memory location accessed and it is shifted appropriately inside the IU. The store address is sent to the data cache.

R (W2) - The IU registers the read data and writes it to the register file. Also 0xffffffff is sent to the data cache along with the appropriate byte marks to complete the store.

R2 - LDSTUB complete.

2.0.4 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:

D - Read operands from register file or bypass from instructions still in the pipe.

E - Do appropriate operation 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).

W - Pipe result into R.

R - Write register file with result.

2.0.5 Integer Multiply

Integer multiply takes 19 cycles to complete. The algorithm implemented in the microSPARC IU is a modified Booth's (2-bit) multiply. The multiply process can be broken up into 4 distinct steps:

Initialization	1 cycle
Booth's iteration	16 cycles

Correction (ala Booth)	1 cycle
Writeback	1 cycle

The first cycle is used to set up the registers used in the multiply. The rs1 and rs2 registers initialize 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 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 register contents are then written to the Y register and the rs2 contents to the destination register in the register file.

2.0.6 Integer Divide

Integer divide takes 39 cycles to complete. If an overflow is detected, the instruction completes in 6 cycles. The algorithm implemented in the microSPARC IU is non-restoring binary division (add and shift). The divide process can be broken into 5 distinct steps:

Divide by zero detection	1 cycle
Initialization/Ovf detection	3 cycles
Non-restoring division iteration	33 cycles
Correction (for non-restoring)	1 cycle
Writeback	1 cycle

Because the microSPARC IU does not allow traps to be taken by help instructions, the first step is to determine if we have 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 register The rs2 operand is passed to the rs2 register (surprise!). The W and rs1 registers are used as accumulators. At the completion of the divide, the W register contains the final quotient.

There are two overflow options for signed divide with a negative result defined in the SPARC Rev 8 manual. The microSPARC IU implements:

$$\text{result} < -2^{31} \text{ with remainder} = 0.$$

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 (add then 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.

2.0.7 CTI's

2.0.7.1 Branches

All branches take a single cycle to execute. There is no penalty for taken vs. untaken branches, even in the event that the instruction previous to the branch sets the condition codes.

In the Decode stage, the IU evaluates the condition codes and branch condition to determine 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.

2.0.7.2 JMPL

JMPL is a two cycle instruction in the microSPARC IU. This is done somewhat uniquely in that there are no help cycles for the JMPL. Instead, there is an interlock that always occurs following the JMPL. This is done to force the IU to fetch the JMPL's delay instruction. In this way, the IU can evaluate whether an RETT is in the JMPL's delay slot and evaluate user/supervisor accesses correctly.

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 cycle.

E - Compute target address and send this to the instruction cache.

W - Not much happens.

R - Write the PC of the JMPL instruction into the destination register.

2.0.7.3 RETT

RETT is a two cycle instruction in the microSPARC IU. Unlike JMPL, the RETT utilizes a help cycle. However, since it must follow an JMPL, the first cycle is always interlocked. This cycle allows the IU to determine that the RETT enters the pipe and can force the correct user/supervisor mode (contained in the PSR.PS bit) for subsequent instruction fetches.

D - Read operands from register file or bypass from instruction still in the pipe. Sign extend immediate operands.

E - Compute target address and send this to the instruction cache.

W - Not much happens.

R - Set PSR.ET to 1, move PSR.PS to PSR.S, and PSR.CWP++.

2.0.7.4 CALL

CALL is a single cycle instruction in the microSPARC IU.

D - Add PC and disp30 to form target address. Send this address to instruction cache. The delay slot fetch starts this cycle.

E - The CALL target is fetched.

W - Not much happens.

R - The PC of the CALL is written to r[15].

2.0.8 Instruction Cache Interface

In the event of an instruction cache miss, the IU will recirculate the missed address to the address bus to the instruction cache and hold the pipeline. Since the miss indication cannot be generated in time to prevent the missed instruction from moving from F to D, the missed instruction is physically in the Decode stage of the pipe.

The instruction cache is implemented so that the missed word of the cache line is returned first. This instruction word is strobed into the Decode stage. The IU is now 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 can use the instructions as soon as the instruction cache receives it. To do this, the IU is told when the instruction addressed by the IU is available to be strobed in. The IU can then selectively hold and release the pipe. One caveat is that the IU must correctly select the address to be sent to the instruction cache (determined by hold).

If one of the instructions encountered during the instruction streaming is a taken CTI whose target is outside of the cache line being filled, the IU can detect this condition (the instruction cache cannot) and hold the pipe.

2.0.9 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 will recirculate the missed address to the data cache address bus and hold the pipeline. Since the data miss indication is not generated in time to prevent the instruction from moving from W to R, the instruction that caused the miss is in its R cycle. Any expected load data must then be directly strobed into the R stage and if the instruction in the E stage expects to get load data (via the load bypass), the load data must also be strobed into the correct E stage register(s).

The data cache is also implemented to return the missed word first. When the data cache indicates that the data is available, the data is passed through the load aligner (for any necessary alignment) and then strobed into the R cycle (and appropriate E cycle) register prior to being written to the register file.

The IU is then free to continue. To limit the complexity of the MMU, however, while the data cache is filling the line, no additional memory operations may be started until the line fill is complete. The exception to this is LDD, as the second word is allowed to be strobed in after the first

2.0.10 Interlocks

2.0.10.1 Load Interlock

There is a single cycle load usage interlock in the microSPARC IU when a load instruction is followed by an instruction that uses the destination register of the load as a source operand.

2.0.10.2 Floating Point Interlocks

There are two types. The first is when the FPU is busy and a new floating point instruction is read into Decode. If the FPU detects a conflict, it will assert the FHOLD signal to prevent dispatch of that instruction until such time that the conflict is resolved.

The second is when a floating point branch enters decode and the FCCV bit from the FPU is deasserted. The interlock persists until the FPU asserts the FCCV bit.

2.0.10.3 Special Register Interlocks

Because of the execute datapath design, the microSPARC IU is unable to bypass special register read data to the instruction immediately following it in the pipeline. A single cycle interlock occurs.

2.0.11 Traps and Interrupts

2.0.11.1 Traps

The microSPARC IU implements all Rev 8 traps except the following optional traps:

data store error

r register access error

unimplemented FLUSH

watchpoint detected

coprocessor exception

Trap priorities are as defined in SPARC Rev 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 due to the presence of the higher priority trap.

In the pipeline, the trap indication usually occurs when the trapping instruction reaches the W stage of the pipeline The exception to this are the exceptions detected by the MMU (e.g.: a LD which causes a data access exception trap) which occur in the MEMOP's R cycle. The reason for this difference is to allow the MMU an additional cycle to determine memory exceptions. Note that traps may be detected as early as the D cycle of the instruction. The trap indication is then piped to the W stage of that instruction.

After the assertion of the TRAP signal, instructions following the trapped instruction in the pipeline are flushed out. The PSR.ET <- 0, PSR.PS <- PSR.S, PSR.S <- 1, and PSR.CWP--. TBR.TT <- trapcode. The PC and nPC are written to r17 and r18. Instruction fetches then transfer operation to the trap vector as defined in the TBR.

The microSPARC IU does not allow help instructions to take a trap There are no deferred integer traps. The IU will detect and act on deferred floating point traps.

2.0.11.2 Interrupts

The microSPARC IU is interrupted via the Interrupt Request Level bus The IU depends on external logic to select the highest priority interrupting device and provide the appropriate IRL level. To discard glitches on the IRL lines, the IU must see at least two cycles where the level on the IRL are the same. Only then does it initiate an interrupt request to the processor. This request is pipelined by one cycle. The interrupt will be taken by the instruction currently in the W cycle of the pipeline (or, if that instruction is a help instruction, by the next non-help W cycle) if the IRL level is greater than the current PIL and there are no higher priority traps that take precedence.

Because there is a one cycle delay between when the IRL and PIL are compared and when the trap priorities are checked, this could cause a problem where back to back PSR writes could cause an interrupt to occur when the existing value in PSR.PIL is greater than the IRL. The microSPARC IU can prevent this from happening in hardware, so we avoid the difficulties encountered with previous designs.

2.0.11.3 Reset Trap

On reset, the following things occur:

- Traps are disabled (PSR.ET <- 0).
- If power-up reset, PSR.PS is undefined, else PSR.PS is unchanged.
- Enter supervisor mode (PSR.S <- 1).
- If power-up reset, PSR.CWP is undefined, else PSR,CWP is unchanged.
- If power-up reset, r[17] and r[18] are undefined, else are unchanged.
- If power-up reset, TBR.TT is undefined, else is unchanged.
- Execution begins at location PC=0 and nPC=4.

2.0.11.4 Error Mode

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

- PSR.S <- 1.
- PSR.PS is unchanged.

- PSR.CWP --
- PC and nPC written to r[17] and r[18].
- PC <- 0, nPC <- 4.
- Assertion of the IU_ERROR signal.

In addition, the TBR.TT may be changed if the trapping instruction is an RETT. The TBR.TT will hold:

- With PSR.S = 0, TBR.TT will reflect privileged instruction
- With a window underflow, TBR.TT will reflect the underflow
- With a misaligned target address, TBR.TT will reflect the misaligned trap.

The IU will remain in error mode until it is reset.

2.0.12Floating Point Interface

The microSPARC IU controls the addresses for all instructions and for floating point memory operations. Within the SingleSparc chip, the floating point unit has its own bus to the instruction cache. The IU provides the necessary strobe to load the FP's instruction register. This includes handling around instruction misses and instruction exceptions. In addition, the IU informs the FPU if the instruction just loaded is valid and should be continued down the pipe.

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 will strobe the FPU's data register to pick up the data when it is available. For floating point stores, the IU starts the cache access and picks up the store data from the FPU. The IU then registers it and provides it to the data cache in the correct cycle(s).

When the FPU detects a usage conflict with the instruction just fetched in Decode, it asserts the FHOLD signal, which causes the IU to *interlock* the pipeline. The interlock is released when the FPU's internal status allows for the new FP instruction to start in the FPU.

FCC and FCCV are used by the IU to determine taken and untaken cases for floating point branches. If a floating point branch is detected in Decode and FCCV is not asserted, the IU will interlock until FCCV is asserted.

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

FPops take one cycle in the IU, plus additional cycles in the FPU. For the number of cycles in the FPU, please refer to the FPU section in this document.

2.0.13 Special Features

The microSPARC IU has some build in features to make debug and bringup easier.

The IU is fully scanned, with all registers connected into the microSPARC IU scan chain (JTAG).

Certain registers of the scan chain are accessible only through the scan chain. These enable certain features useful for bringup and debug.

RF bypass - each read port has a bypass enable that causes the write data to be bypassed to the read port. Two registers in the scan chain can be set to enable this. These registers will be zeroed immediately on the next clock (when scan mode is off), disabling this feature.

Illegal opcode event - when this feature is enabled through the scan chain, the IU will assert the iu_event signal when a certain illegal opcode is decoded in the pipeline and the instruction causes an illegal instruction trap The opcode in question is op=10 binary and op3=111111 binary. Once enabled, this feature can only be cleared through the scan chain.

IU error event - when this feature is enabled through the scan chain, the IU will assert the iu_event signal when the IU enters error mode. Once enabled, this feature can only be cleared through the scan chain.

2.0.14 Divergence from SPARC version 8

The microSPARC IU has been designed to SPARC version 8 compatible (as currently defined in the SPARC Architecture Manual, Version 8, Review-2) including hardware integer multiply and divide. microSPARC IU does deviate from full support of version 8 features due to system design criteria. The deviations are as follows.

The microSPARC IU PSR is as implemented in the SPARC Rev 8 manual. In early specifications of the microSPARC IU, it was stated that the EC bit of the PSR is not writeable. To maintain compatibility with Rev 8 and IEEE 1754, the PSR.EC bit *is* writeable. Rev 8 states that Coprocessor disabled traps occur when a coprocessor instruction is decoded and PSR.EC=0 *or a coprocessor is not present.*

Alternate space memory operations proceed normally, however with a single caveat. Rather than the 8 bits of ASI, the microSPARC MMU only decodes 6 bits. The IU was directed to drop these bits, so out of bound ASI encodings are not detected.

The microSPARC IU does not implement STBAR since there is no need to force store ordering in this system. It will pass through the pipe as a Read Y Register operation with destination being the bit bucket (%g0).

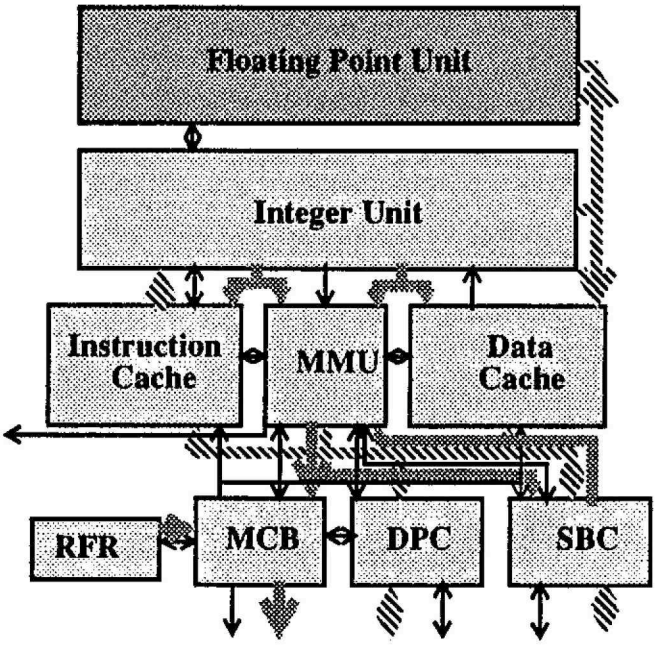
The microSPARC IU also does not support reads and writes to the any Modes or Ancillary State Registers. We have no need for these. All read

cases will act like a Read Y Register operation. All write cases will act like a NOP.

The value read from the implementation field (IMPL) of the PSR for Tsunami will be (hexadecimal) 4. The value read from the version field of the PSR will be (hexadecimal) 1.

3.0 Floating Point Unit

The microSPARC Floating Point Unit is based on the Meiko FPU design. The Meiko FPU has been tailored for low-cost, and matched to the SPARC IU to balance performance. The FPU performance is more a result of the data cache hit rate, than the peak performance provided by the FPU design. The performance is therefore based on system level modeling, including the appropriate cache hit rates.

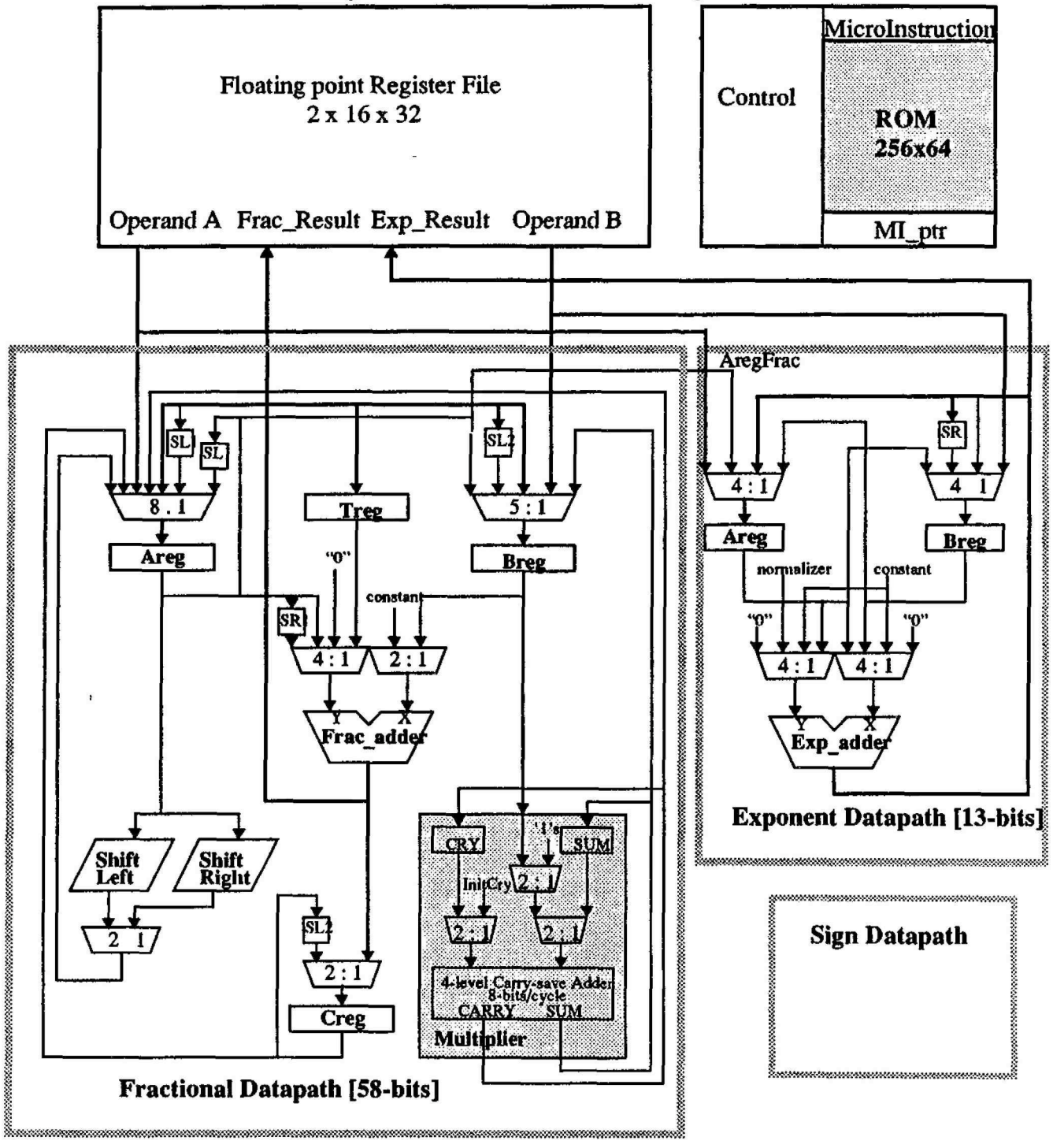


3.0.1 Overview

The Meiko FPU design is based on matching performance with the SPARC integer unit. The match is achieved by examining the maximum bandwidth of the integer unit in starting floating point operations and executing FPU LOADs and STOREs.

The Meiko FPU fully executes all single and double precision FP instructions as defined in the SPARC Architecture Manual (Version 8), except fsmuld. All other FP instructions (including fsmuld) trap to unimplemented. All implemented instructions will complete in hardware, therefore this FPU will never generate an unfinished exception. A block diagram follows:

Figure 3.0 - Meiko FPU Block Diagram



The bandwidth of the caches and main memory, and the integer unit's ability to fetch operands and schedule floating point instructions is the bottom line in performance. Through simulation, it has been determined

that the IU cannot provide data and schedule FP instructions at a rate faster than about 6 cycles per flop. The Meiko FPU can sustain floating operation times of about 5 cycles (as seen in LINPACK traces), and therefore will hardly impact overall operation time compared to an infinitely fast FPU.

The above conclusions allow a FPU implementation using multiple cycles to complete complex operations. The following algorithms were chosen for their positive trade-off in contributing to the final size and speed of the FPU.

- 8-bit multiply step
- 2-bit division step
- 1-bit square root step
- short distance (0-15 bits) shifter/normalizer
- separate single cycle round step
- microcode state machine to control FPU and decode operation

3.0.2 Deltas from SPARC version 8

The microSPARC FPU deviates from SPARC version 8 by not supporting the fsmuld instruction or quad-precision floating-point operations, and traps to unimplemented when these instructions are encountered. The microSPARC FPU also differs from the Appendix N, "SPARC IEEE 754 Implementation Recommendations" NaN format. The following figure shows the value returned for an untrapped floating-point result in the same format as the operands:

Figure 3.1 - Untrapped FP Result in Same Format as Operands

		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

In the figure above, all QNaN results will have their sign bit set to 0 ME_NaN is 0x7fff0000 (single-precision) or 0x7fffe00000000000 (double-precision).

Figure 3.2 - Untrapped FP Result in Different Format

	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**

In the figure above, +imax = 0x7fffffff, and -imax = 0x80000000. (QNaN2) is a copy of the mantissa bits of the operand, with the extra low order bits zeroed, and the sign bit zeroed.

3.0.3 Implementation Specific Features

The microSPARC FPU implements a 1-entry floating-point deferred trap queue. When a floating-point instruction generates an fp_exception, microSPARC will delay the taking of an fp_exception trap until the next floating-point instruction is encountered in the instruction stream. The microSPARC FPU implementation can be modeled as having 3 states: fp_execute, fp_exception_pending, and fp_exception. These are shown in the figure below.

Normally the FPU is in fp_execute state. It moves from fp_execute to fp_exception_pending when an FPop generates a floating-point exception.

The FPU moves from fp_exception_pending to fp_exception, when the IU attempts to execute any floating-point instruction (including fbcc's). This transition (FXACK) generates an fp_exception trap. At this time the FQ contains the instruction and address of the FPop which originally caused the fp_exception.

An fp_exception trap can only be caused while the FPU is moving from the fp_exception_pending state to the fp_exception state (or by executing a STDFQ instruction when FSR.qne == 0, as described below). While in fp_exception state, only floating-point store instructions may be executed (particularly STDFQ and STFSR) and they can not cause an fp_exception trap.

The FPU remains in the fp_exception state until a STDFQ instruction is executed and the FQ becomes empty. At that time, the FPU returns to the fp_execute state.

If an FPop, or a floating-point load instruction (excluding fbcc's and all store instructions) is executed while the FPU is in fp_exception state, the FPU returns to fp_exception_pending state and also sets the FSR.ftt

field to sequence_error (0x4) The instruction that caused the sequence_error is not entered into the FQ.

If a STDFQ instruction is executed when the FQ is empty (FSR.qne == 0, FPU is in fp_execute state), the FPU will generate an immediate fp_exception trap (not deferred) and set the FSR.ftt field to sequence_error (0x4), but the FPU will remain in fp_execute state.

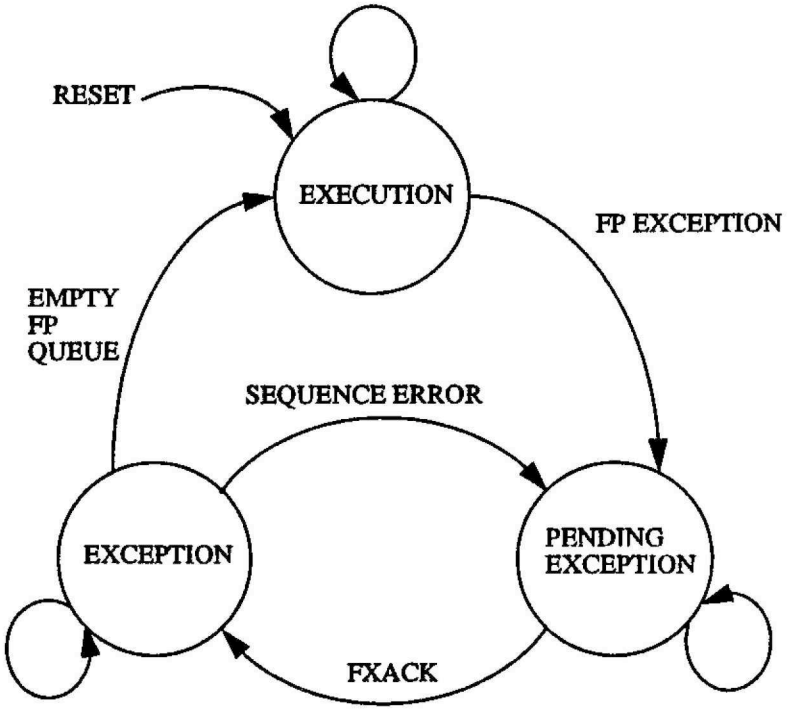


Figure 3.3 - FPU Operation Modes

The STDFQ instruction will store the address from the FQ to the effective address, and the instruction from the FQ to the effective address + 4.

3.0.4 Software Considerations

This section describes the software visible features of the microSPARC FPU/FPC.

The FSR.ftt field is set whenever an FPop completes or causes an exception. This field will remain unchanged until another FPop completes (or causes a sequence error). The FSR.ftt field may be cleared by executing a non-trapping FPop, such as fmovs%f0,%f0.

The following table describes the bits in the Floating-Point Status Register (FSR):

Table 3.0 - FSR Summary

FSR Bits	field	values	Description	writeable by LDFSR
31:30	RD	0 - Round to nearest (tie-even) 1 - Round to zero 2 - Round to +infinity 3 - Round to -infinity	Rounding Direction	Yes
29:28	res	always 0	reserved	No
27:23	TEM	0 - disables corresponding trap 1 - enables corresponding trap	Trap Enable Mask	Yes
22	NS	always 0	Nonstandard FP	No
21:20	res	always 0	reserved	No
19:17	ver	always 4	FPU version number	No
16:14	FTT	0 - None 1 - IEEE Exception 2 - Unfinished FPop 3 - Unimplemented FPop 4 - sequence error	FP trap type	No
13	QNE	0 - queue empty 1 - queue not empty	Queue Not Empty	No
12	res	always 0	reserved	No
11:10	FCC	0 - == 1 - < 2 - > 3 - ? (unordered)	FP Condition Codes	Yes
9:5	AEXC	0 - no corresponding exception 1 - corresponding exception	Accrued Exception Bits	Yes
4:0	CEXC	0 - no corresponding exception 1 - corresponding exception	Current Exception Bits	Yes

3.0.5 FPU Instruction Timings

The instruction timings, as quoted by Meiko, are provided in the following table. The timings are in CPU cycles.

Table 3.1 - FPU Instruction Timings

Instruction	Min	Typ	Max
fadds	4	4	17
faddd	4	4	17
fsubs	4	4	17
fsubd	4	4	17
fmuls	5	5	25
fmuld	7	9	32
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	4	15
fcmpd	4	4	15
fcmpes	4	4	15
fcmped	4	4	15
unimplemented	3	3	3

These cycle counts assume that the operands are available in the register file. A load-use interlock (fp load followed by an FPop which uses the destination register of the load as an operand) may add up to 2 cycles to the typical cycle count.

Because of the limited shifter size (0-15 bits was chosen to save hardware), the fpu instruction cycle counts are data dependent. There are 5 ways in which operations may take longer than the typical cycle count:

1. 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 will be trapped (if traps are enabled).

2. Possible exceptional results (results which are very close to underflow or overflow) may add up to 5 cycles to the typical cycle count. In a normal environment these are rare events, probably caused by ill-conditioned data.

3. Denormalized operands will add 1 extra cycle for each 15 bit shift required to normalize before the operation, and 1 extra cycle for each 15 bit shift required to denormalize the result after the operation (if necessary). Because operations on denormalized numbers will always complete in hardware (this fpu will never generate an unfinished exception), the overall performance will be greater than for an fpu which traps on denormalized operands.

4. Add or Subtract which require an initial alignment of more than 15 bits will add 1 extra cycle for each 15 bit shift. Also, a Subtract result which requires a shift of more than 15 bits to normalize will add 1 extra cycle for each 15 bit shift.

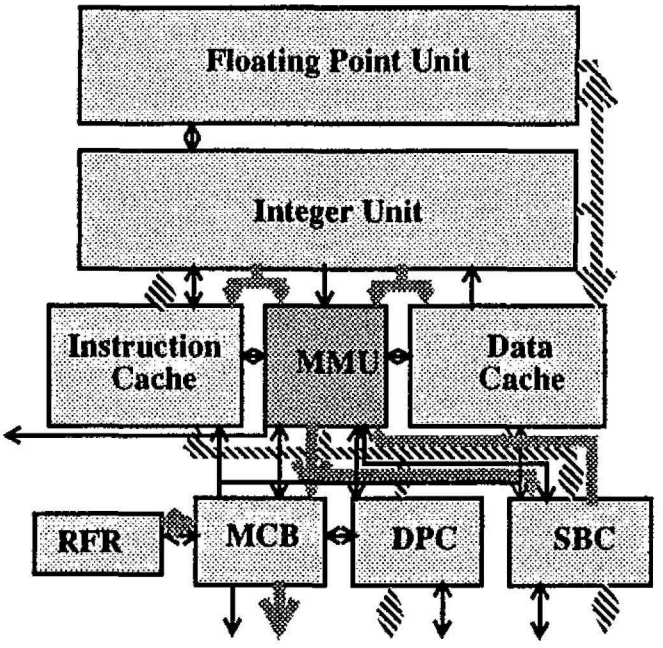
5. Non-standard rounding modes (RZ and RN are the typical operating modes) may require up to 3 additional cycles for some corner cases and exceptions.

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

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

4.0 Memory Management Unit

The microSPARC MMU provides the functionality of both a reference MMU as specified by the SPARC Reference MMU Architecture and a Sun 4M IO MMU. Additionally, much of the memory arbitration logic is contained within the MMU block.

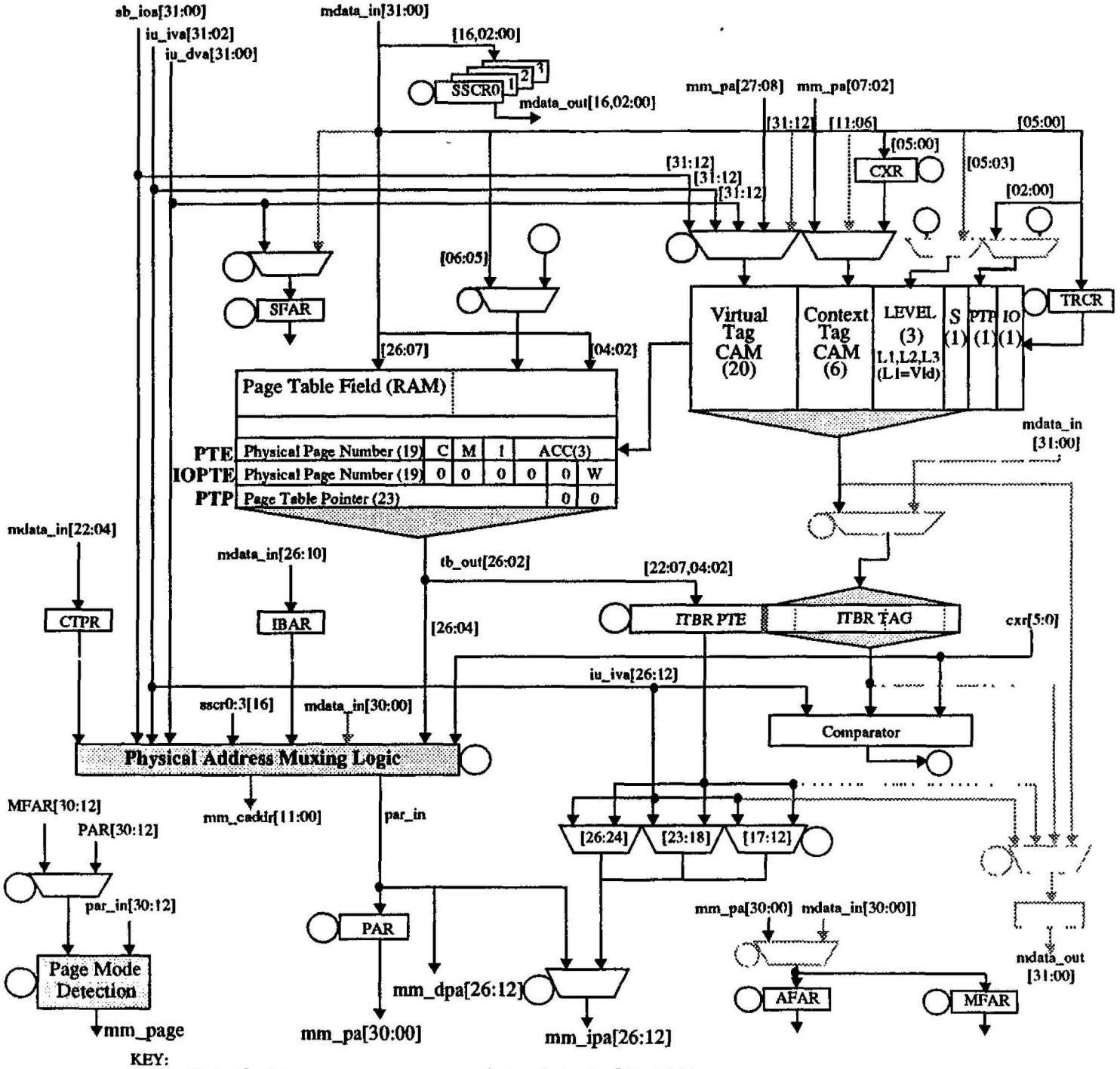


4.0.1 Overview

This MMU provides four primary functions. First, the MMU translates virtual addresses of each running process to physical addresses in memory. More specifically, the MMU provides translation from a 32 bit virtual address to a 31 bit physical address by using a translation lookaside buffer (TLB). The 3 high order bits of Physical Address are maintained to support memory mapping into 8 different address spaces. The MMU supports the use of 64 contexts Second, the MMU provides memory protection so that a process can be prohibited from reading or writing the address space of another process. Page protection and usage information is fully supported. Third, the MMU implements virtual memory. The page tables are maintained in main memory. When a miss occurs in the TLB the table walk is handled in hardware and a new virtual to physical address translation is loaded into the TLB. Finally, the MMU performs the arbitration function between IO, Data Cache, Instruction Cache, and TLB references to memory.

The reference MMU contains a 32 entry fully associative TLB and uses a pseudo random algorithm for the replacement of TLB entries. An address and data path block diagram follows.

Figure 4.0 - MMU Address and Data Path Block Diagram



KEY:

CXR - Context Register
CTPR - Context Table Pointer Register
PAR - Physical Address Register
ITBR - Instruction Translation Buffer Register
SSCR - SBus Slot Configuration Register
SFAR - Synchronous Fault Address Register
AFAR - Asynchronous Fault Address Register
MFAR - Memory Fault Address Register
IBAR - IOMMU Base Address Register
TRCR - TLB Replacement Control Register

iu_iva - Instruction Virtual Address
iu_dva - Data Virtual Address
sb_ioa - IO Address
bd_mdata - 32 Bit Internal Memory Bus
mm_pa - Physical Address (to SBC, MCB)
mm_ipa - Instruction Physical Address (to ICache)
mm_dpa - Data Physical Address (to DCache)
mm_caddr - CAS address bits to MCB
tb_out - output from TLB RAM (note that the verilog implements these as 24:00 not 26:02)

○ - From/To State Machine or Control Logic
~~~ - Diagnostic use
~~~

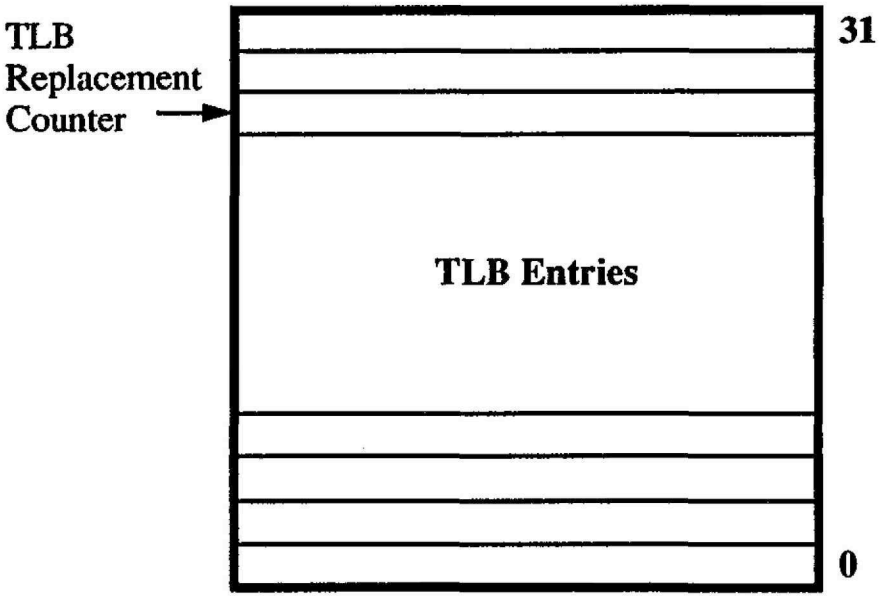
4.0.2 Translation Lookaside Buffer

The TLB is a 32 entry, fully associative cache of page descriptors It caches 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 generically describe any virtual address (sb_ioa, iu_iva or iu_dva) or physical address (mm_pa, mm_dpa or mm_ipa) respectively.

4.0.2.1 TLB Replacement

The TLB uses a pseudo random replacement scheme. There is a 5 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. A simple diagram follows.

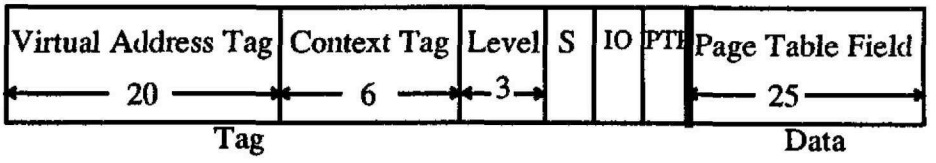
Figure 4.1 - TLB Replacement



4.0.2.2 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 4.2 - TLB Entry



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 when referencing PTEs and IOPTEs. VA[11:00] is the byte within a page. The address in this field is physical when referencing PTPs with the least significant 19 bits containing PA[26:08].

Context Tag - The 6 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] in order to have a TLB hit. This field contains a physical address (PA[07:02]) when referencing PTPs. This field is not used when referencing IOPTEs.

Level - The 3 bit level field is used to enable the proper virtual tag match of region, and segment PTE's. IOPTE's and PTP's will have this field set to use Index 1, 2 and 3 (b'111'). The most significant bit also serves as the TLB Valid bit because it is set for any valid PTE, IOPTE, or PTP. The following table defines the level field:

Table 4.1 - Virtual Tag Match Criteria

Level	Match Criteria
000	None
100	Index 1 (VA[31:24])
110	Index 1, 2 (VA[31:18])
111	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 is a supervisor level (ACC=6 or 7).

IO Page Table Entry (IO) - This bit indicates that an IOPTE resides in this entry of the TLB.

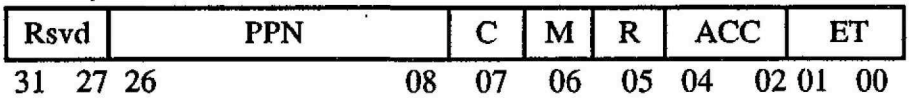
Page Table Pointer (PTP) - This bit indicates that a PTP resides in this entry of the TLB. Note that all SRMMU flush types (except page) will flush all PTPs from the TLB.

Page Table Field - The page table field can either be a Page Table Entry (PTE), a Page Table Pointer (PTP), or an IO Page Table Entry (IOPTE). This field can be read and written using ASI 0x06.

4.0.2.3 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 4.3 - Page Table Entry in Page Table



Field definitions:

Reserved (Rsvd) - Bits [31:27] should be written as zero, and will be read as zero.

Physical Page Number (PPN) - This field is 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 follows.

Table 4.2 - Page Table Access Permissions

ACC	Permissions	
	User	Supervisor
0	Read only	Read only
1	Read/Write	Read/Write
2	Read/Execute	Read/Execute
3	Rd/Wrt/Exec	Rd/Wrt/Exec
4	Execute only	Execute only
5	Read only	Read/Write
6	No access	Read/Execute
7	No access	Rd/Wrt/Exec

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. The bit definitions of the ET field follows:

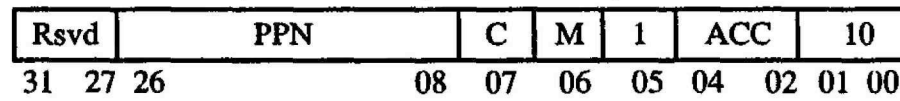
Table 4.3 - Page Table Entry Types

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 RAM the PTE has the following format:

Figure 4.4 - Page Table Entry in TLB



Bits [31:27] are not implemented, should be written as zero, and will be read as zero.

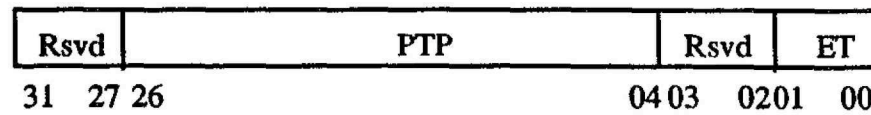
Bit [05] is set to one by hardware indicating that every PTE in the TLB has been referenced.

Bits [01:00] are set to one:zero 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

4.0.2.4 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 field in a PTP tag is always set to 0x7. A PTP is defined as follows:

Figure 4.5 - Page Table Pointer in Page Table



Field definitions:

Reserved (Rsvd) - Bits[31:27,03:02] should be written as zero, and will be 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 true of the context table at the root level also. The sizes of the tables are summarized as follows.

Table 4.4 - Sizes of Page Tables

Level	Size (Bytes)
Root	256
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. The bit definitions of the ET field follows:

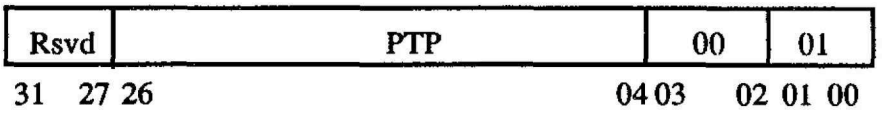
Table 4.5 - Page Table Entry Types

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 following format:

Figure 4.6 - Page Table Pointer in TLB



Bits [31:27] are not implemented, should be written as zero, and will be read as zero.

Bits [03:02] are set to zero by hardware and are unused.

Bits [01:00] are set to zero:one by hardware indicating the entry type (ET) of a PTP. These bits are not actually stored in the TLB rather are derived as a function of the PTP bit of the tag.

4.0.2.5 IO MMU Page Table Entry

An IO Page Table Entry (IOPTE) defines both the physical address of a page and its access permissions. Note that the Level field in a IOPTE tag is always set to 0x7 and the Supervisor bit is set to 0x0. An IOPTE is defined as follows.

Figure 4.7 - IO Page Table Entry in Page Table

Field definitions:

Reserved (Rsvd) - Bits [31:27] are not implemented, should be written as zero, and will be read as zero. Bits [07:03] should also be written as zero, and will be read as zero.

Physical Page Number (PPN) - This field is the high order 19 bits of the 31 bit physical address of the page. The PPN appears on PA[30:12] when a translation completes. This address is concatenated with VA[11:00] to provide the entire translated address.

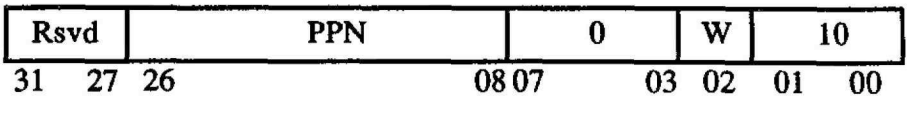
Writeable (W) - When this bit is set to a one both reads and writes to the page are allowed. When this bit is zero only reads are allowed.

Valid (V) - This bit is set to a one when the IOPTE is valid.

Write As Zero (WAZ) - This bit is to be written as zero in the memory io pagetable by software.

In the TLB an IOPTE has the following format:

Figure 4.8 - IO Page Table Entry in TLB



Bits [31:27] are not implemented, should be written as zero, and will be read as zero.

Bits [07:03] are set to zero by hardware. Bit[05] is used to distinguish between PTEs (set to 1) and IOPTEs (set to 0). Bits[07:06,04:03] are unused.

Bits [01:00] are set to one:zero by hardware indicating a valid IOPTE These bits are not actually stored in the TLB.

4.0.3 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 follows:

Table 4.6 - Virtual Tag Match Criteria

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

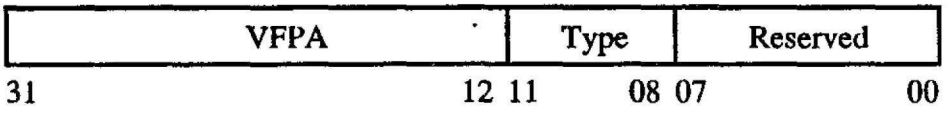
In addition to the virtual tag match, context matching of a PTE is required for all user page references (ACC is 0 to 5) when made by either user or supervisor (ASI = 0x8-0xB). Context matching is not required for a supervisor page reference (ACC is 6 or 7) when made by a supervisor (ASI = 0x9 or 0xB). 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.**

4.0.4 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 4.9 - CPU TLB Flush or Probe Address Format



Field Definitions:

Virtual Flush or Probe Address (VFPA) - This field is the address that is used to index into TLB. Depending on the type of flush or probe not all 20 bits are significant.

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.

4.0.4.1 CPU TLB Flush

The flush operation must remove the PTEs and PTPs from the TLB that match the type criteria as follows:

Table 4.7 - TLB Entry Flushing

Type	Flush	PTE Match Criteria
0	Page	(Level 3) AND (Context match OR ACC=6-7) AND VA[31:12] match
1 to 4	Entire	None (Entire TLB Flush)
5 to F	Reserved	

4.0.4.2 CPU TLB Probe

The probe operation returns either a PTE from a page table in main memory or the TLB or it returns 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 were needed into the TLB. If the PTE was not already there the referenced bit is updated. Probe type 0 affects

one entry of the TLB which is invalidated at the end of the probe operation.

Probe types 1-3 should be preceded by a TLB Flush Entire to ensure correct operation

Table 4.8 - CPU TLB Entry Probing

Type	Probe	Returned Data
0	Page	Level 3 PTE or 0
* 1	Segment	Level 2 PTE or 0
* 2	Region	Level 1 PTE or 0
* 3	Context	Level 0 PTE or 0
4	Entire	PTE from Table Walk or 0
5 to F	Reserved	

* - Must be Preceded by TLB Flush Entire

4.0.5 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 and 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 0x4. The address map for these registers follows.

Table 4.9 - Address Map for MMU Registers

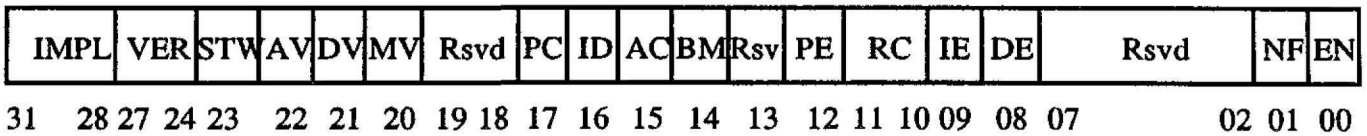
VA[12:08]	Register
00	Processor Control Register
01	Context Table Pointer Register
02	Context Register
03	Synchronous Fault Status Register
04	Synchronous Fault Address Register
05-0F	Reserved
10	TLB Replacement Control Register
11-12	Reserved
13	Synchronous Fault Status Register**
14	Synchronous Fault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

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 register description for details.

4.0.5.1 Processor Control Register

The Processor Control Register contains control and status bits for the microSPARC processor. The BM, IE, DE, and EN bits receive both the sbus reset (normal reset) and watchdog resets (BM is set, IE, DE, and EN are reset). It is highly recommended that sta's to the PCR are immediately followed by a SPARC FLUSH instruction to keep the machine in a very consistent state. The PCR is defined as follows:

Figure 4.10 - Processor Control Register



Field Definitions:

Reserved (Rsvd) - Bits [19:18,13,07:02] are unimplemented, should be written as zero and will be read as zero.

Implementation (IMPL) - The implementation number of this SPARC Reference MMU. This field is hardwired to 0x4 and read only.

Version (VER) - The version number of this SPARC Reference MMU. This field is hardwired to 0x1 read only.

Software Tablewalk enable (STW) - This bit enables the instruction_access_MMU_miss and data_access_MMU_miss traps for instruction and data tablewalking respectively for tablewalks to be done by software.

Address View (AV) - This bit is used for diagnostic purposes. Any address from the MMU Physical Address Register (PAR) is displayed on the SBus Address pins (SBADDR[27:00 = mm_pa[27:00]). This is a debug and test feature. During debug this can be monitored while running non io diagnostics. You cannot use the sbus while this bit is set.

Data View (DV) - This bit is used for diagnostic purposes. Any Data on the internal memory data bus will appear on the external SBus Data pins (SBDATA[31:00]). This is a debug and test feature. During debug this can be monitored while running non io diagnostics. You cannot use the sbus while this bit is set.

Memory Data View (MV) - This bit is used for diagnostic purposes. Any Data on the internal memory data bus (mdata[31:00]) will appear on the external memory data pins. This is useful for monitoring ASI and control space accesses (from/to both the IU and SBus). You cannot get to memory when this bit is set for either load or store operations.

Refresh Control (RC) - These 2 bits control the DRAM refresh rate of the system. Normal 40MHz operation would require a 0x2 value. The RC field is defined as follows:

Table 4.10 - Memory Refresher Control Definition

RFR_CNTL	Refresh Interval
0	Every 128 clocks (to 8.6 MHz)
1	No Refresh
2	Every 512 clocks (to 35 MHz)
3	Every 768 clocks (to 52 MHz)

Parity Control (PC) - This bit controls the generation of parity (and checking on memory reads) in the memory interface as follows:

Table 4.11 - Parity Control Definition

PC	Meaning
0	Even Parity
1	Odd Parity

ITBR Disable bit (ID) - This bit **disables** the use of the Instruction Translation Buffer Register when set.

Alternate Cacheability (AC) - This bit specifies that the caches are enabled by the IE and DE bits even with the mmu disabled when set. When not set, the caches are disabled when the mmu is disabled. This should not be used during boot mode accesses (or other instruction accesses to an sbus device).

Boot Mode (BM) - This bit is set by both sbus 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 non video 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 zero, all references miss the cache. This bit is reset by both sbus reset and watchdog reset.

Data Cache Enable (DE) - The data cache is enabled when this bit is set to a one When zero, all references miss the cache. This bit is reset by both sbus reset and watchdog reset.

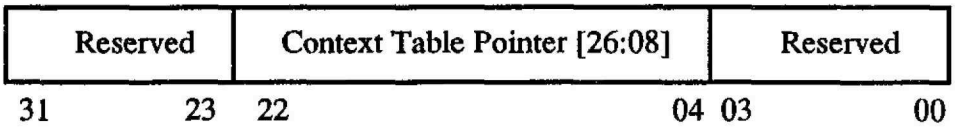
No Fault bit (NF) - When set the supervisor accesses which cause exceptions will not be signaled to the processor (will be captured in the SFSR). Normal operation occurs while this bit is cleared.

MMU Enable (EN) - When this bit is set to a one the MMU is enabled and translation occurs normally. When this bit is not set the physical address is forced to the 31 least significant bits of the virtual address. This bit is reset by both sbus reset and watchdog reset.

4.0.5.2 Context Table Pointer Register

The Context Table Pointer Register (CTPR) contains the base of the Context table. It is defined as follows.

Figure 4.11 - Context Table Pointer Register



The Context Table Pointer is 19 bits wide. The reserved fields are unimplemented, should be written as zero, and read as a zero.

4.0.5.3 Context Register

The Context Register (CXR) is used as an index into the Context table. It is defined as follows.

Figure 4.12 - Context Register

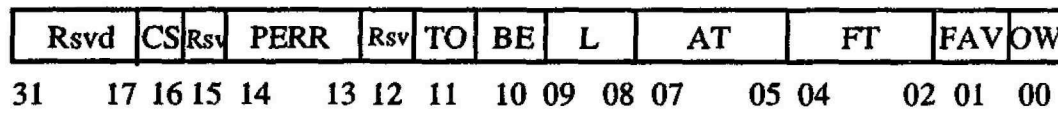
The Context Register defines which virtual address space is considered the "current" address space. Subsequent accesses to memory through the MMU are translated for the current address space. This continues until the CXR is changed. The physical address of the root pointer is obtained by taking bits [22:04] from the CTPR to form mm_pa[26:08] and bits [05:00] from the CXR to form mm_pa[07:02]. mm_pa[30:27,01:00] are zero. Bits [31:06] of the CXR are unimplemented, should be written as zero, and read as a zero.

4.0.5.4 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 type 0x03 clears it. Using type 0x13 to read the SFSR 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 actually signalled. In other words, it may not be valid if there is no exception

Figure 4.13 - Synchronous Fault Status Register



Field Definitions:

Reserved (Rsvd) - 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: [1] invalid ASI space, [2] invalid ASI size, [3] invalid VA field in valid ASI space and [4] 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.

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.

Sbus Time Out (TO) - An Sbus Time Out resulted from a CPU initiated read transaction. No Sbus slave responded with an acknowledge within 256 Sbus cycles (12.8 us).

Sbus Bus Error (BE) - An error indication was returned from an Sbus slave on a CPU initiated read transaction. This may have been either an error acknowledgment or a late error.

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 as follows

Table 4.12 - 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 as follows. Note that this field is not valid on Control Space Errors.

Table 4.13 - 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 as follows.

Table 4.14 - 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 as follows.

Table 4.15 - Setting of SFSR Fault Type Code

AT	FT Code								
	PTE[V]=0	PTE[V]=1							
		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	-

An invalid address error code (FT=1) is set when an invalid PTE or PTP is found while fetching an entry from the page table for a regular table walk or a probe entire operation. 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, or on a synchronously generated SBus error acknowledge or time out. 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).

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 faults and data translation 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.

Table 4.16 - 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

If a single access causes multiple errors, the fault type is recognized in the following priority.

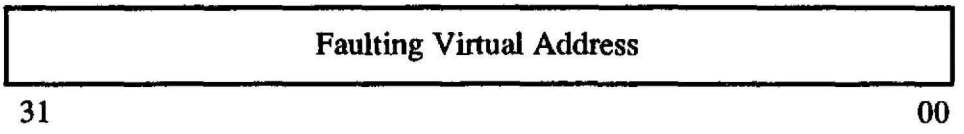
Table 4.17 - Priority of Fault Types on Single Access

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

4.0.5.5 Synchronous Fault Address Register

The Synchronous Fault Address Register (SFAR) records the 32 bit virtual address of any data fault 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 that the SFAR should always be read before the SFSR to insure that a valid address is returned.** The structure of this register is as follows.

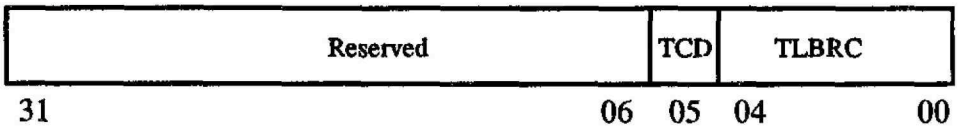
Figure 4.14 - Synchronous Fault Address Register



4.0.5.6 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.

Figure 4.15 - TLB Replacement Control Register



Field Definitions:

Reserved - Bits [31:06] are unimplemented, should be written as zero and will be read as zero.

TLB Replacement Counter Disable (TCD) - The TLBRC will 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 TCD bit is set When a TLB miss occurs, the counter value is used to address the entry to be replaced.

4.0.6 IO MMU Registers

The IO MMU Control Register (IOCR) contains IO MMU control and status flags. The IO MMU Base Address Register (IOBAR) defines the base address of the IO PTE Table in memory The SBus Slot Configuration Registers (SSCR[0:3]) provides information about the slave device in the spare SBus slots. If a parity error occurs on an IO initiated transaction the physical address causing the fault is placed in the Asynchronous Fault Address Register (AFAR) and the cause of the fault can be determined from the contents of the Asynchronous Fault Status Register (AFSR). A DMA parity error will result in asserting the level 15 interrupt output (to be fed back to the IU externally as an interrupt) and the assertion of an error acknowledge to the SBC so it can return an SBus error acknowledge to the device that initiated the

transaction IOPTE entries may be flushed from the TLB by doing writes to the Address Flush Register (AFR). This register is write only. All of these internal MMU registers can be accessed directly by software using SBus and IO MMU Control Space accesses with PA[30.24]=0x10. Also, the Entire TLB can be flushed using a control space access. The SBus and IOMMU Control Space address map follows.

Table 4.18 - SBUS and IO MMU Control Space

PA<30:00>	Device	R/W
1000 0000	IO MMU Control Register	R/W
1000 0004	IO MMU Base Address Register	R/W
1000 0014	Flush All TLB Entries	W
1000 0018	Address Flush Register	W
1000 1000	Asynchronous Fault Status Reg.	R/W
1000 1004	Asynchronous Fault Address Reg.	R/W
1000 1010	SBUS Slot Configuration Register0	R/W
1000 1014	SBUS Slot Configuration Register1	R/W
1000 1018	SBUS Slot Configuration Register2	R/W
1000 101C	SBUS Slot Configuration Register3	R/W
1000 1020	Memory Fault Status Register	R/W
1000 1024	Memory Fault Address Register	R/W
1000 2000	MID Register	R/W

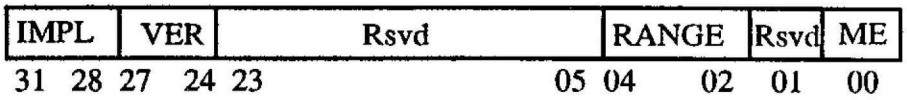
4.0.6.1 IO MMU Control Register

The IO MMU Control Register (IOCR) contains control and status bits for the IO MMU This register can be accessed using Sbus and IO MMU Control Space (0x10000000).

NOTE: Control space loads should not be executed while DMA is enabled (see MID register). A possible deadlock condition may occur if a DMA atomic or quad-word write coincides with the control space load.

The IOCR is defined as follows:

Figure 4.16 - IO Control Register



Field definitions:

Implementation (IMPL) - The implementation number of this IO MMU. This field is hardwired to 0x4 and read only.

Version (VER) - The version number of this IO MMU. This field is hardwired to 0x1 and read only.

Reserved (Rsvd) - Bits [23:05,01] are not implemented, should be written as zero, and will be read as zero.

RANGE - This field defines the virtual address range for DVMA. Specifically, the translatable limit is defined to be 16MB*2**<RANGE>. All VA bits above this limit must be set to one for an address to be valid. For example, if RANGE=2 then 64MB of virtual address are supported, and valid DVMA virtual addresses range from 0xFC000000 to 0xFFFFFFFF. Any access using a DVMA virtual address that is out of that range will receive an SBus error acknowledge. The only exception involves slots that have Bypass Enabled. The following table shows how the physical address of an IO MMU page table entry is generated:

Table 4.19 - IO MMU Page Table Address Generation

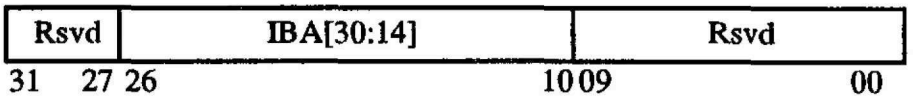
Range	Limit	Physical Address[30:00]
0	16MB	IBAR[26:10], IOVA[23:12],b'00'
1	32MB	IBAR[26:11], IOVA[24:12],b'00'
2	64MB	IBAR[26:12], IOVA[25:12],b'00'
3	128MB	IBAR[26:13], IOVA[26:12],b'00'
4	256MB	IBAR[26:14], IOVA[27:12],b'00'
5	512MB	IBAR[26:15], IOVA[28:12],b'00'
6	1GB	IBAR[26:16], IOVA[29:12],b'00'
7	2GB	IBAR[26:17], IOVA[30:12],b'00'

IO MMU Enable (ME) - IO MMU translation is enabled when this bit is set.

4.0.6.2 IO MMU Base Address Register

The IO MMU Base Address Register (IBAR) defines base address of the IO Reference Table. This register can be accessed using Sbus and IO MMU Control Space (0x10000004). The IBAR is defined as follows.

Figure 4.17 - IO MMU Base Address Register



Field definitions:

Reserved (Rsvd) - Bits [31:27,09:00] are not implemented, should be written as zero, and will be read as zero.

IO MMU Base Address (IBA) - When the IO MMU is enabled and the access translation misses the TLB, IBA is used as the base address for the (<RANGE/1024>)byte-aligned IO MMU Reference Table.

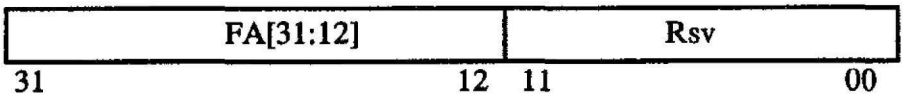
4.0.6.3 IOMMU Flush All TLB Entries

All TLB entries are flushed by writing to control space address PA=0x10000014. This address should not be read since the output of the TLB is unknown during a flash clear operation.

4.0.6.4 IOMMU Address Flush Register

The IOPTE entries may be flushed from the TLB by doing writes to the Address Flush Register at PA=0x10000018 with the following format. The Address Flush Register is defined as follows.

Figure 4.18 - IOPTE Address Based Flush Format



Field definitions:

Reserved (Rsv) - Bits [11:00] are not implemented and should be written as zero.

Flush Address (FA) - The virtual page address of the IOPTE entry to be flushed.

Note that a register is not actually implemented to perform this function. Also note that to flush all IOMMU entries all TLB entries must be flushed (see section on CPU TLB Flush for details).

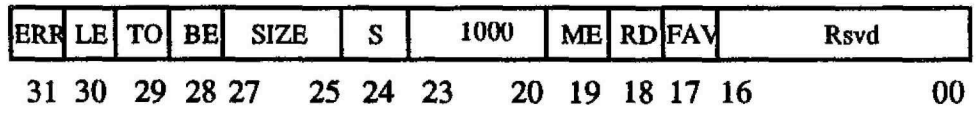
4.0.6.5 Asynchronous Fault Status Register

The Asynchronous Fault Status Register (AFSR) provides information on asynchronous faults during IO initiated transactions and CPU write operations. This register is used only for PIO operations, and is accessed using Sbus and IO MMU Control Space (0x10001000). A hardware lock is used to ensure that this register does not change while being read. Reading this register clears it. Multiple errors set the ME bit, but do not change any other states. The AFSR always reflects the status of the first error. Refer to the Sun 4M specification.

Note: The AFSR.size field is invalid when a late error (AFSR.le) is detected.

Note: Due to the pipelined nature of Processor I/O space writes, it is possible to receive a late error (AFSR.le) and no longer have the correct address stored in the AFAR. When this occurs, the AFSR.fav bit will not be asserted, indicating that the AFAR contains an invalid address.

Figure 4.19 - Asynchronous Fault Status Register



Field Definitions:

Reserved (Rsvd) - Bits [23:20,16:00]. Bits [23:20] are forced to '1000'. Bits [16:00] are not implemented, should be written as zero, and read as zero.

Summary Error Bit (ERR) - One or more of LE, TO, or BE is asserted.

Late Error (LE) - The SBus reported an error after the transaction was done.

Time Out (TO) - An SBus write access timed out.

Bus Error (BE) - An SBus write access received an error acknowledge.

Size (SIZE) - SBus size of error transaction.

Supervisor (S) - CPU was in Supervisor mode when error occurred.

Multiple Error (ME) - At least one other error was detected after the one shown.

Read Operation (RD) - The error occurred during a read operation.

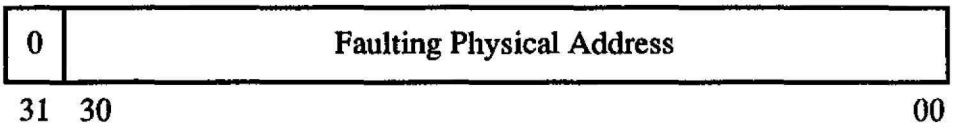
Fault Address Valid (FAV) - The address contained in the AFAR is accurate and can be used in conjunction with the status in AFSR. The only time the AFAR will be invalid is on an SBus late error in which the second processor IO operation has already been requested and is queued up in the SBC.

4.0.6.6 Asynchronous Fault Address Register

The Asynchronous Fault Address Register (AFAR) records the 31 bit physical address that caused the fault. This register is accessed using Sbus and IO MMU Control Space (0x10001004). Bit [31] should be written as zero and will be read as zero. A hardware lock is used to

insure that this register does not change while being read. Writing the AFSR unlocks the AFAR. The structure of this register is as follows.

Figure 4.20 - Asynchronous Fault Address Register

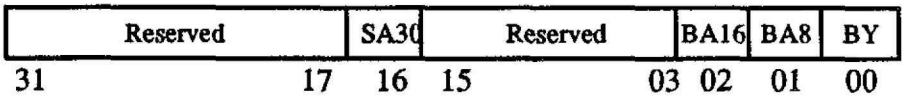


Note that bit 31 is unimplemented, should be written as zero, and will be read as zero. Also, this register is only held when an error is reflected in the AFSR.

4.0.6.7 SBUS Slot Configuration Registers

The SBus Slot Configuration Registers (SSCR[0:3]) provide information about the slave device in sbus slots, and is also used for IO MMU bypass management for that slot. These registers can be accessed using Sbus and IO MMU Control Space (0x10001010, 0x10001014, 0x10001018 and 0x1000101C respectively).The SSCR is defined as follows:

Figure 4.21 - SBUS Slot Configuration Register



Field definitions:

Reserved - Bits [31:17,15:03] are not implemented, should be written as zero, and will be read as zero.

Segment Address Bit 30 (SA30) - This bit provides PA[30] when IO MMU bypass is used.

BA16 - Slave supports 16 byte bursts.

BA8 - Slave supports 8 byte bursts.

IO MMU Bypass (BY) - When this bit is set the MMU is bypassed and the virtual addresses from this slave are treated as physical when sb_ioa[31:30]=00. mm_pa[30] is given by the SA30 field and mm_pa[29:00] is defined as sb_ioa[29:00].

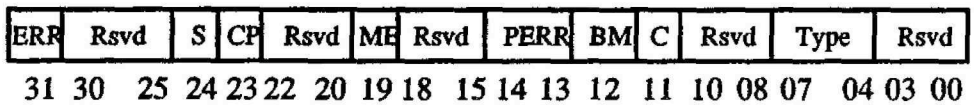
4.0.6.8 Memory Fault Status Register

The Memory Fault Status Register (MFSR) provides information on parity faults. This register is accessed using Sbus and MMU Control Space (0x10001020). 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 was an error condition. Reading this register allows it to begin loading once again.

When multiple memory errors occur, the MFSR will hold the status reflecting the operation in which the first error occurred, and also set the multiple error bir (MFSR.me). The MFSR will maintain the error status until cleared, which can be done by reading the MFSR.

Figure 4.22 - Memory Fault Status Register



Field Definitions:

Reserved (Rsvd) - Bits [30:25,22:20,18:15,10:08,03:00] 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 error occurred.

CPU Transaction(CP) - CPU initiate 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).

Boot Mode (BM) - This bit indicates that the error occurred while the PCR was indicating that we were in Boot Mode.

Cacheable (C) - Address of error was mapped cacheable. On a CPU initiated transaction this bit is 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 follows:

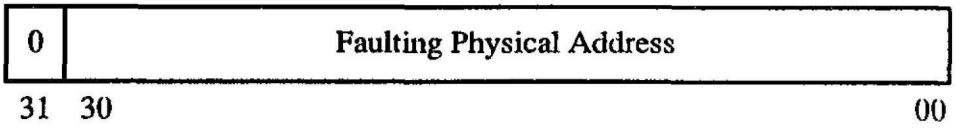
Table 4.20 - Memory Request Type

Value (Hex)	Name	Meaning
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		Reserved
6		Reserved
7		Reserved
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		Reserved
E		Reserved
F		Reserved

4.0.6.9 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 Sbus and IO MMU Control Space (0x10001024). 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 was an error condition. Reading this register allows it to begin loading once again. Bit [31] should be written as zero and will be read as zero The structure of this register is as follows.

Figure 4.23 - Memory Fault Address Register



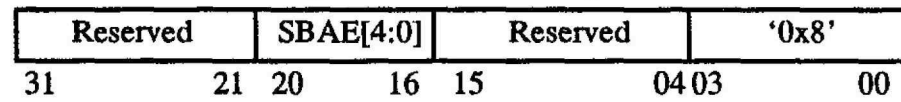
Note that bit 31 is unimplemented, should be written as zero, and will be read as zero. Also, this register is only held when an error is reflected in the MFSR.

4.0.6.10 MID Register

The MID Register contains two fields. The MID field (Bits[3:0] contain a constant value of 0x8) and the SBAE field which controls the ability

of SBus devices to arbitrate for the bus. This register can be accessed using Sbus and IO MMU Control Space (0x10002000).The SBAE bits are both readable and writeable while the MID field is read only. The MID is defined as follows:

Figure 4.24 - MID Register



Field definitions:

Reserved - Bits [31:21,15:04] are not implemented, should be written as zero, and will be read as zero.

SBus Arbitration Enable[4:0] (SBAE) - These bits control the ability for devices on the SBus to arbitrate for the bus. The most significant bit (SBAE[4]) controls arbitration for the SCSI/ Ethernet master. The other bits (SBAE[3:0]) control arbitration for SBus devices 3:0 corresponding to SSCR[3:0]. These bits are R/W.

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

4.0.7 IO MMU Bypass Mode

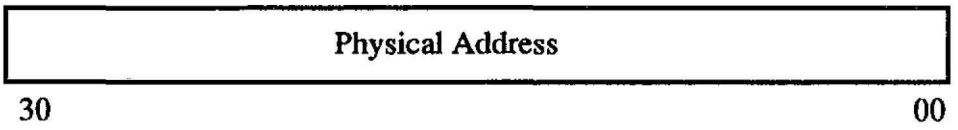
Bypass mode is provided to allow intelligent SBus masters to do their own memory management with assistance from the kernel This facility is enabled by having the Bypass Enable bit set in that device's slot configuration register. It is assumed that such a master will have its own MMU. In order to bypass the IO MMU the DVMA master must issue a virtual address with sb_ioa[31:30]=0. In this case the Physical Address bus will have the Virtual Address bus put on it. The PA is checked to verify that it is in the valid main memory range and an error is issued to the master if it is not.

4.0.8 Physical Address Register

The Physical Address Register (PAR) is used to hold translated physical addresses before they are used for either memory requests or for Sbus

operations. This register cannot directly be read or written. The structure of this register is as follows.

Figure 4.25 - Physical Address Register

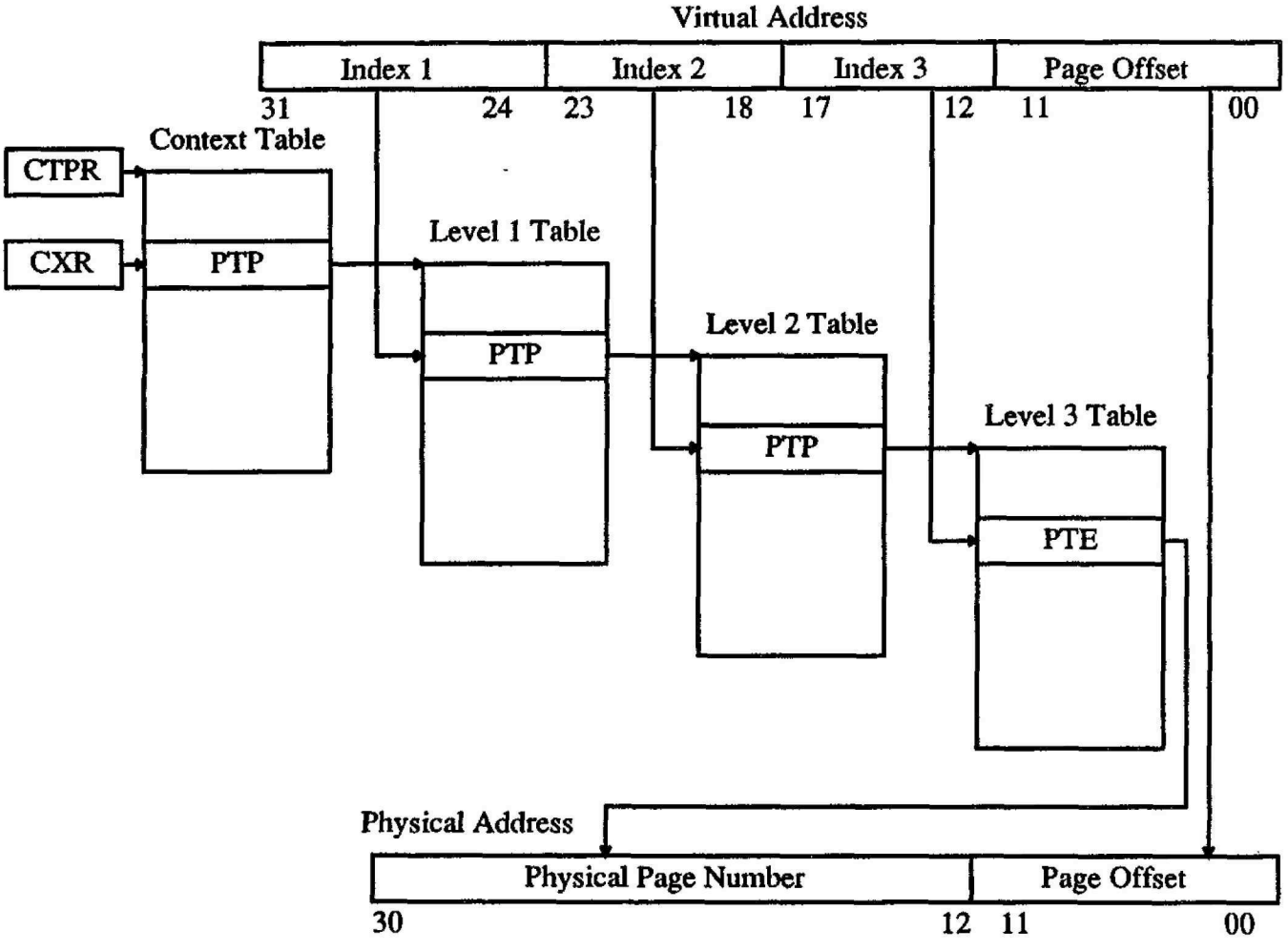


4.0.9 TLB Table Walk

On a translation miss the table walk hardware translates the virtual address to a physical address by "walking" through a context table and from 1 to 3 levels of page tables. The first and second levels of these tables typically (not necessarily) contain page table pointers (PTP) to the next level tables when accesses are due to CPU instruction or data addresses. IO accesses only the first level page table. A third level table entry should always be a page table entry (PTE) pointing to a physical page or else a translation fault occurs.

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 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 the following figure.

Figure 4.26 - CPU Address Translation Using Table Walk



When the PTE is found it is stored in an available TLB entry and used to complete the original virtual to physical address translation. A table walk which was 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 had not been already.

The table walk for an IO generated virtual address uses the IO Base Address Register (IOBAR) as a base register and part of the DVMA virtual address as an index into an IOPTE table in memory. Specifically the IO MMU page table size and corresponding DVMA virtual address range are configured in the IOCR RANGE field. The table consists of 4 byte entries. The virtual address used for this mapping is VA[X:0] where "X" is the highest VA bit in the translatable range. VA[31:X+1] must be all "1"s in order for translation to take place; otherwise an error is signalled to the DVMA master. The bits VA[X:12] provide a virtual

page number which is used as an index into the IOMMU table in memory. These bits are placed on mm_pa[X-10:2]. The rest of the physical address is mm_pa[1:0] = 00, and mm_pa[30:X-9] = IBA[30:X-9]. This is the PA used for the one level IO walk.

4.0.10 Instruction Translation Buffer Register

Since instruction fetches occur every time the pipeline moves and there is only one TLB for translating instruction references, data references and DVMA requests, a method for dealing with conflicts between instruction references and data or IO references to the TLB was needed A registered version of the last instruction translated TLB line is kept in the Instruction Translation Buffer Register (ITBR). When the TLB arbiter determines there is a conflict the iu_iva goes to the ITBR and the two translations occur simultaneously. When the iu_iva misses in the ITBR the translation is done in the TLB the next available cycle Note that the default is to translate instruction addresses in the TLB and the ITBR is used only for conflict cases. This maximizes the hit rate of instruction address lookups. Each time an iu_iva is successfully translated in the TLB the ITBR is updated. The ITBR is logically split into a PTE and Tag section. Both the PTE and tag portions of the ITBR are read and written like other TLB PTE and tags using ASI 0x6. See diagnostic section for details.

An I-cache miss will require a translation using the TLB, as there is no datapath from the ITBR to PAR. Therefore, the ITBR is only useful for cached pages.

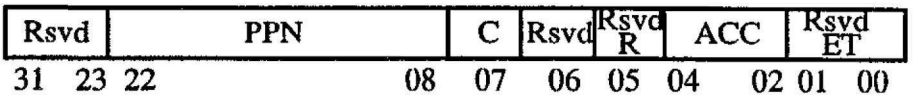
Any access error detected by the ITBR is seen as an ITBR miss, without updating any Fault Status logic. Normal execution will retry the translation using the TLB, and set the Fault Status logic accordingly.

The ITBR is invalidated whenever the TLB is written or flushed to maintain consistency. The ITBR is always a copy of the TLB entry, not an additional entry.

4.0.10.1 ITBR Page Table Entry

An ITBR Page Table Entry (ITBR/PTE) defines both the physical address of a page and its access permissions. A ITBR/PTE is defined as follows.

Figure 4.27 - ITBR Page Table Entry



Field definitions:

Reserved (Rsvd) - Bits [31:23,06:05,01:00] are not implemented, should be written as zero, and will be read as zero except for bits [05 and 01] which are read as one. This was done to make the ITBR appear as a valid PTE when read Bit [06] is the M bit (=0), bit [05] is the R bit (=1) and bits [01:00] is the ET field (=10 for PTE).

PPN, C, ACC - these fields are defined the same as they for TLB PTEs. Note that the 4 most significant PPN bits are not kept in the ITBR since instruction references must be made to main memory (limit 128MB in address space 0).

4.0.10.2 ITBR Tag

An ITBR Tag is defined in the section on MMU diagnostic strategy. Briefly, the tag consists of the Level field, the Instruction Virtual Address Tag, and the Context Tag.

4.0.11 Arbitration

The MMU block performs the primary memory arbitration function on the CPU. This is due 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), the data cache block (for loads and stores), the TLB (during tablewalks and to keep the referenced and modified bits in the main memory page tables up to date), and IO 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 out of the MMU.

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

4.0.11.1 TLB Arbitration

The current priority scheme places TLB references as highest priority, followed by IO references, data references, and finally instruction references. Note that the TLB is referenced during every CPU clock in normal operation. 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 4.21 - TLB Reference Priority

Operation Pending			
IO DMA	IU Data Ref.	Instr. Fetch	Result
YES	X	X	Xlate for IO, Tablewalk if miss, use ITBR for IFetch Xlate
NO	YES	X	Xlate for IU Data Reference, Tablewalk if miss, use ITBR for IFetch Xlate
NO	NO	YES	Xlate for Instruction Fetch, Tablewalk if miss, load ITBR with Xlate output

Note: X=Don't Care, Xlate=Translate

4.0.12 Translation Modes

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

Table 4.22 - Translation Modes

Name	ASI	Boot Mode	MMU En.	PA[30:00]
Boot IFetch	0x8, 0x9	Yes	Off	PA[30:28]=0x7, PA[27 00]=VA[27:00]
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]

4.0.13 Page Mode Detection

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 or not. This is done by comparing the contents of the MFAR (at the time of the last request) with the current physical address (mm_pa) the cycle before a request is ready. Specifically, bits [26:12] have to match between MFAR and the PA If these bits match then the MMU will assert PAGE. The memory controller then has the option of using a page mode DRAM access or not. If mm_page is not asserted then a page mode access cannot be used to fulfill the request.

4.0.14 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. In a Sun4M system, this would indicate a level 15 interrupt.

4.0.15 Diagnostic Features

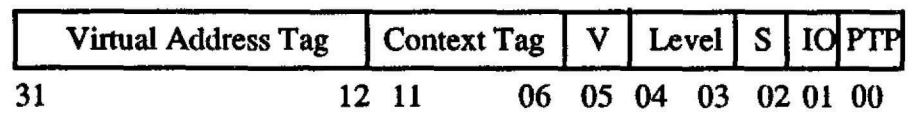
All registers and RAM (and CAM) are accessible directly through alternate virtual address space loads and stores. In addition to this control is provided for putting the internal memory data bus onto the external memory data or SBus data pins. Also, any generated physical address can be seen at the SBus address pins.

There is also the ability to breakpoint on certain conditions This is set up through use of the scan chain. More details follow.

4.0.15.1 Diagnostic Access of TLB, ITBR

Diagnostic reads and writes to the 32 TLB entries and the ITBR are performed by using load and store alternate instructions in ASI 0x6 and the virtual address to explicitly select a particular TLB entry. The access must be a word access, all other data sizes will result in an internal error. Depending on the virtual address specified either the TLB Tag, TLB PTE, ITLB Tag or ITLB PTE will be referenced. The format for the TLB PTE is as described earlier. The format of the Tag is shown below: (Note that bits [02:00] are not valid for an itbr tag and are read as zero)

Figure 4.28 - CPU Diagnostic TLB and ITLB 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 PTPs with the least significant 19 bits containing PA[26:08].

Context Tag - The 6 bit context tag comes from the value in the context register as 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]) when referencing PTPs

Valid bit, Level bits - These 3 bits are used to enable the proper virtual tag match of root, region, and segment PTE's. The Valid bit indicates a valid entry.

Supervisor (S) - This bit is used to disable the matching of the context field indicating that a page is a supervisor level (ACC=6 or 7). This bit is non meaningful for an ITLB Tag and is read as 0.

IO Page Table Entry (IO) - This bit indicates that an IOPTE resides in this entry of the TLB. This bit is non meaningful for an ITLB Tag and is read as 0.

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) will flush all PTPs from the TLB. This bit is non meaningful for an ITLB Tag and is read as 0.

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 2 PTE for example. The TLB output is not valid when this occurs.

Note: Any sta to the TLB tag or data must be followed by 3 nops. This is to allow the pipelined TLB write sufficient time to complete.

The virtual address is used to select the TLB entries as follows:

Table 4.23 - TLB Entry Address Mapping

Virtual Address	TLB Entry
0x0	Entry 0 PTE
0x4	Entry 0 Tag
0x8	Entry 1 PTE
0xC	Entry 1 Tag
0x10	Entry 2 PTE
0x14	Entry 2 Tag
0x18	Entry 3 PTE
0x1C	Entry 3 Tag
0x20	Entry 4 PTE
0x24	Entry 4 Tag
0x28	Entry 5 PTE
•	•
•	•
•	•
0xF0	Entry 30 PTE
0xF4	Entry 30 Tag
0xF8	Entry 31 PTE
0xFC	Entry 31 Tag
0x100-0x7FC	Reserved
0x800	ITBR PTE
0x804	ITBR Tag
0x808-FFFFFFFC	Reserved

4.0.15.2 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. The basic idea is to stop the clocks when certain conditions occur. This facility is general purpose in that there is a large matrixed selection of conditions to choose from. The breakpoints which can be enabled are virtual address matching, virtual address source matching, virtual address type matching, memory request matching, tablewalk detection (includes type), and tablewalk level matching. A more detailed description and suggested pairings of these conditions follows.

We have the ability to breakpoint on portions of the virtual address (the output of the virtual address muxing logic). The ITBR must be turned off to guarantee matches on instruction addresses. These portions of the virtual address can be combined with other conditions to make their match conditions more case specific as follows:

Table 4.24 - Virtual Address Match Conditions

Virtual Address Conditions	Conditions to be Paired With
VA[31:00]	Any address translation:
VA[31:01]	io_tlb (DMA read, write or translate)
VA[31:02]	dc_tlb (iu load, store, or atomic op)
VA[31:03]	ic_tlb (instruction translation)
VA[31:12]	or
VA[31:18]	The following cycle types:
VA[31:24]	read_w (iu load in w stage)
VA[10:02]	write_w (iu store in w stage)
VA[11:02]	ldsto_w (iu atomic in w stage)
!VA[31:12] & VA[11:02]	iu_fetch_f (instr. fetch in f stage)
!VA[31:11] & VA[10:02]	sb_read (DMA read op)
	sb_write (DMA write op)
	sb_translate (DMA translate - before DMA write op)

The virtual address breakpoint control register enables specific address bits for comparison. The details of the register are listed below:

Table 4.25 - Virtual Address Field Enable Decode

10	9	8	7	6	5	4	3	2	1	0
I 31:24	H 23:18	G 17:12	F 11	E 10:04	D 03	C 02	B 01	A 00	N11 N11	N NOT

Enables A-I 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=1, normal comparisons are made. When N=0, the compare result is inverted; so, a 'hit' occurs when the addresses mismatch. The same control applies to N11. As an example, to enable the address bits VA [31:00}, as listed in table 2.4.24, a value of 0x7FF is required in the virtual address breakpoint control register.

We also have the ability to breakpoint on the particular type of memory request being sent from the MMU to the MEMIF. This is sampled when a memory request is actually being issued (mm_issue_req = 1). This 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 follows:

Table 4.26 - Memory Request Type

Memory Request	
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
io_tlb_tw	Tablewalk from DVMA
Tablewalk Level	
Root Level	
Level 1	
Level 2	
Level 3	

4.0.15.3 Additional Features

There are other features which can be used for microSPARC debug.

Some of these features are enabled using Processor Control Register bits. Software tablewalks can be enabled by asserting PCR[23], the STW bit. When in this mode the mmu will cause the instruction_access_MMU_miss and data_access_MMU_miss traps for instruction and data tablewalking respectively for tablewalks to be done by software.

The view modes are also very useful features for both debug and vector generation. There are three view modes: Address View, Data View, and Memory Data View which are enabled by PCR[22:20] respectively.

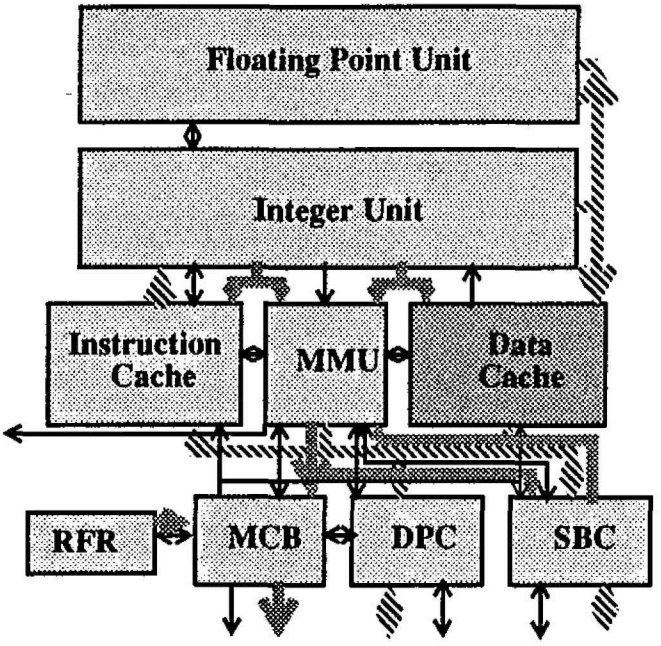
Address View mode is useful for non io testing allowing the Physical Address Register (PAR) to be viewed (1 cpu cycle later) on the SBus Address lines (bits [27:00] only).

Data View mode is useful for non io testing allowing the internal mc_mdata tristate bus to be viewed (1 cpu cycle later) on the SBus Data lines (bits [31:00]).

Memory Data View is useful for non memory sequences allowing the internal mc_mdata tristate bus to be viewed (1 cpu cycle later) on the Memory Data lines (bits [31:00]).

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 (or other instruction accesses to an sbus device). Specifically, having the mmu off, instruction cache on, alternate cacheability on and an sbus instruction access can cause indeterminate data to be put into the instruction cache. Instruction accesses work fine with alternate cacheability when the accesses are to main memory space.

5.0 Data Cache

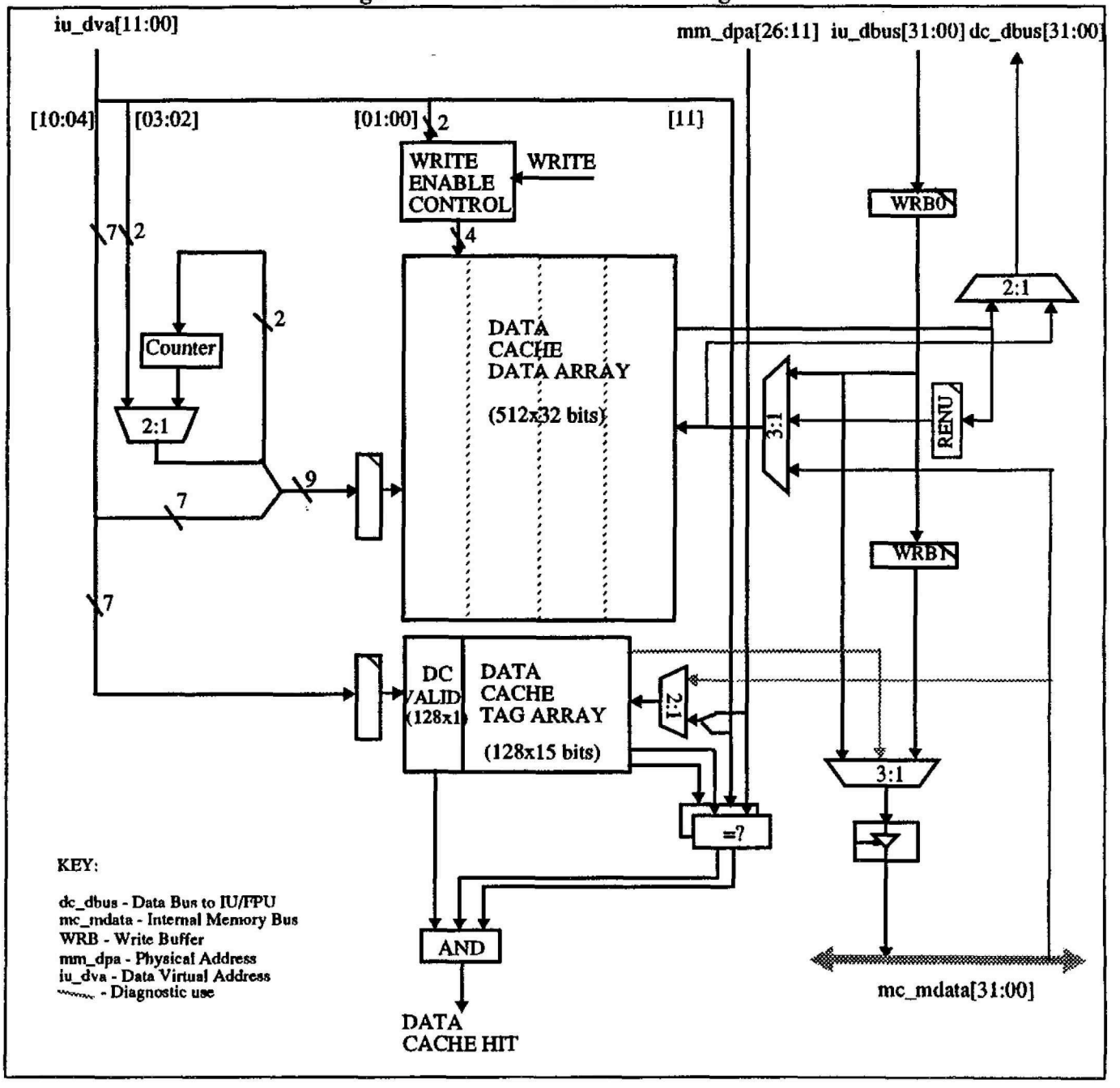


5.0.1 Overview

The microSPARC Data Cache is a 2K-Byte, direct mapped cache, used on load or store accesses from the CPU to cacheable pages of main memory. It is virtually addressed but physically tagged. Stores are write-through with no write allocate. The data cache is addressed by iu_dva[10:0]. The data cache is organized as 128 lines of 16 bytes of data. Each line has a cache tag store entry associated with it. 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 also cache bypass paths. These paths are used for noncached load references, and for streaming data into the IU or FPU on cache miss. A simple block diagram follows.

Figure 5.1 - Data Cache Block Diagram



5.0.2 Data Cache Data Array

All IU write operations to cached locations write the data through to main memory, i.e. on a write hit, both the data cache and main memory are updated. There is, however, no write allocate, i.e. no cache fill is done on a write miss.

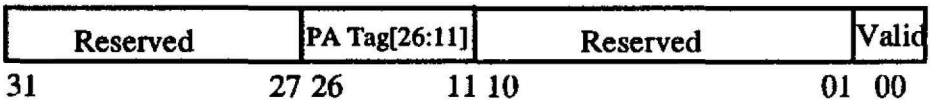
System software may read and write the data cache directly by executing load or store alternate space instructions, of any size, in ASI 0xF. Virtual address bits [10:0] will be used to address the data cache in this mode; all other virtual address bits are ignored during these operations.

There are three input sources to the data cache data array. The IU data_out bus (iu_dbus) is used when the data cache is updated on an integer or floating-point store operation. The internal memory data bus (mc_mdata) is used as input for fills on data cache misses. The RENU register is used in cancelling writes on stores which miss the cache.

5.0.3 Data Cache Tags

A data cache tag entry consists of several fields as follows.

Figure 5.2 - Data Cache Tag Entry



Field Definitions:

Reserved (Rsvd) - Bits [31:27,10:01] are not implemented, should be written as 0 and will be read as 0.

Physical Address Tag - This field contains the physical address of the data held in the cache line. The Data Cache Controller writes this field from bits [26:11] of the physical address (mm_dpa) of the line.

Valid - This bit indicates that the line contains data. This bit is set when a cache line is filled due to a successful cache miss; a cache fill which results in a memory parity error will leave the Valid bit unset. An alternate address space data cache flash clear operation will clear the valid bits of all of the data cache tag entries.

There are two input sources to the data cache tag array. The Physical Address bits needed for the tag are used for cache updates due to data cache misses. The internal memory data bus (mc_mdata) is used as input for alternate store operations.

System software can read and write the data cache tags by executing word-length LDA and STA (Load and Store Alternate) instructions in ASI 0xE. The Virtual address bits [10:4] will select one of the 128 tags; all other address bits are ignored.

5.0.4 Write Buffers

The Write Buffers (WRB0,WRB1) are 32-bit registers in the data cache block used to hold data being stored from the IU or FPU to memory or other physical devices. On a store operation of a word or less, WRB0 holds the store data until it has been sent over the mc_mdata bus to the destination device. For halfword or byte stores, this data is left-shifted (with zero-fill) into proper byte alignment for writing to a word-addressed device before being loaded into WRB0. On a doubleword store the even word is first placed into WRB0. The next cycle the data from WRB0 is moved to WRB1 and WRB0 is loaded with the odd word These registers can be read using a word-length LdA in ASI 0x39; for this operation, bit 8 of the Virtual address selects between the two registers (0 for WRB0, 1 for WRB1).

5.0.5 Data Cache Fill

The memory block size of data fetched from memory on data cache misses is 16 bytes. Memory will always return 16 bytes of data starting with the requested word first followed by the other word of the first doubleword and continuing with another doubleword (even word, then odd) which will wrap around a 16 byte boundary until the entire 16-byte block has been returned. The transfer rate is two words every three cycles from memory (two words of a doubleword, then a dead cycle). The Cache array is loaded the cycle that each word appears on the mc_mdata bus. The following table illustrates the fill operation showing the order that words are written into the cache:

Table 5.1 - Data Cache Fill Ordering

Requested Word	Order of fill (modulo 16B)
0	0, 1, dead cycle, 2, 3
1	1, 0, dead cycle, 2, 3
2	2, 3, dead cycle, 0, 1
3	3, 2, dead cycle, 0, 1

During cache fill, data is bypassed (or "streamed") into the IU or FPU as it is written into the cache data array. For misses on word, halfword, or byte loads, the requested word is bypassed to the IU or FPU in the

same cycle that it appears on the mc_mdata bus; for LdD misses, each of the two requested words is bypassed to the IU or FPU in the same cycle that it appears on the mc_mdata bus.

5.0.6 Internal Memory Bus Interface

The data cache block interfaces to the internal memory bus (mc_mdata). Data from the data cache block to mc_mdata comes from either WRB0 or WRB1. WRB1 is used only for StD and ASI reads of WRB1. There are control signals from the MMU and Memory Controller to indicate when data is on mc_mdata to be loaded into the Data Cache and when data from WRB0 or WRB1 is to be put onto mc_mdata.

5.0.7 IU Data Bus Interface

The data cache block interfaces to an input and output IU data bus (iu_dbus and dc_dbus). Data to the IU or FPU is sourced from either the mc_mdata bus (for streamed data on data cache misses, and for non-cached loads) or the data cache (for data cache hits). Data from the IU or FPU on store operations is always loaded into WRB0.

5.0.8 RENU Register

In the event of a data cache miss on a Store instruction, the cache miss indication is not available until sometime into the cycle in which the store data is being written into the data array. This is too late to inhibit the write operation, so, to prevent the cache line from being corrupted by this write, we use the miss indication to MUX onto the cache array data-in bus a copy of the previous contents of the cache data array location being written. The previous contents of each stored-to location is captured in a special 32-bit register during the tag check access cycle which immediately precedes the write cycle of each store instruction. This register is known as the "REstore if Not Updated" (RENU) Register.

5.0.9 Data Cache Flushing

The data cache is implemented with a flash clear mechanism that is activated by any type of alternate store instruction to ASI 0x37. All data cache valid bits are reset (to zero) by this operation. Note that the data cache is not flushed by the FLUSH instruction (the instruction cache is)

5.0.10 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 operations are not cached.

Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU CR=0)

Accesses while the data cache is disabled (DE bit of the MMU CR=0).

Accesses while using the MMU bypass ASI (ASI=0x20) and alternate cacheability is disabled (AC bits of the MMU CR=0).

Accesses while in Boot Mode.

Accesses to sources in physical address spaces 1-7.

Accesses by the MMU during tablewalks.

5.0.11 Diagnostic Strategy

Sublines and cache tags may be both read and written using ASI 0xF and 0xE respectively as previously discussed. The data cache will be structurally tested via the JTAG controller test ports. All register bits within the data cache and data cache tag are accessible via scan; on the chip level, all locations of these RAMs may be read or written by appropriate sequences of scan operations.

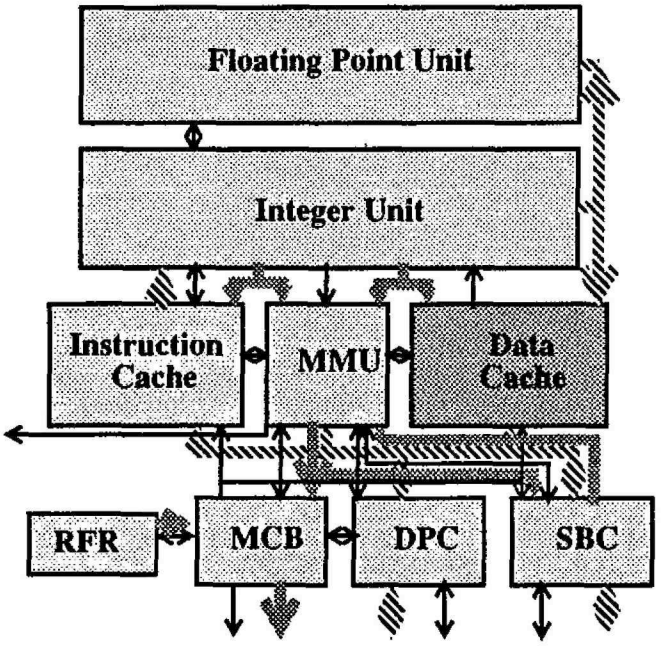
The internal Data Cache Registers may be read using ASI 0x39 and the Virtual Address to reference them. Single word accesses only should be used, others result in an internal error. The Virtual Address map to these registers:

Table 5.2 - Address Map for Data Cache Registers

VA[08]	Register
0	Write Buffer 0
1	Write Buffer 1

iu_dva bits [31:09,07:00] are ignored and should be set to zero by software.

6.0 Instruction Cache



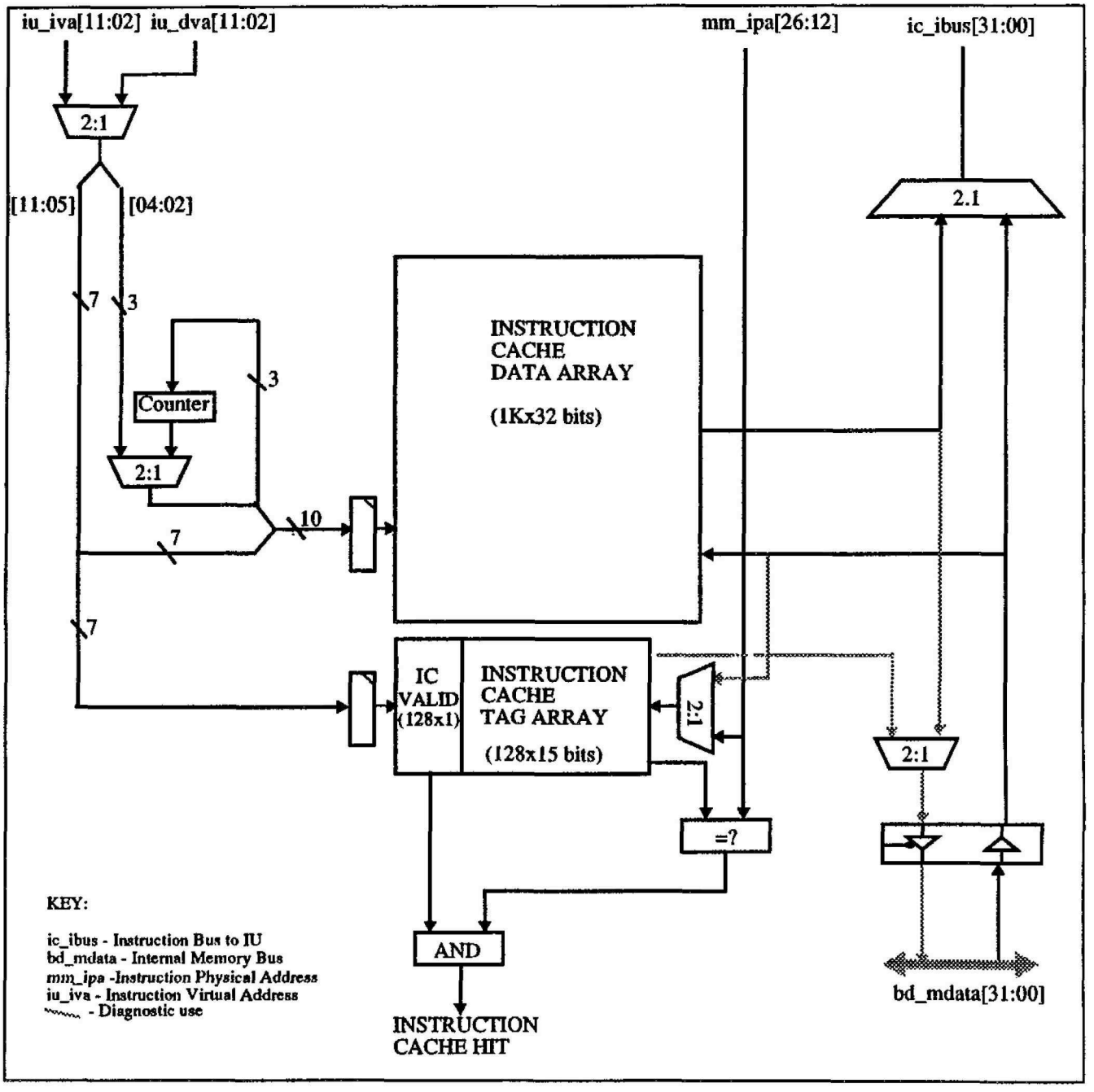
6.0.1 Overview

The microSPARC Instruction Cache is a 4K-Byte, direct mapped cache, used on instruction fetch accesses from the CPU to cacheable pages of main memory. It is virtually addressed but physically tagged. The instruction cache is normally addressed by iu_iva[11:0]. The instruction cache is organized as 128 lines of 32 bytes of data. Each line has a cache tag store entry associated with it. On a instruction cache miss to a cacheable location, 32 bytes of data are written into the cache from main memory.

Within the instruction cache block there are also cache bypass paths These paths are used for noncached instruction fetches, and for

streaming instructions into the IU on cache miss. A simple block diagram follows.

Figure 6.1 - Instruction Cache Block Diagram



6.0.2 Instruction Cache Data Array

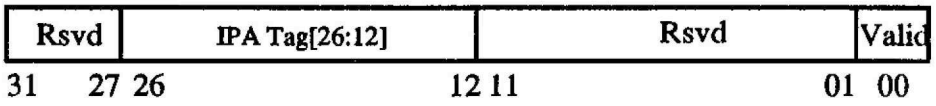
System software may read and write the instruction cache directly by executing load or store word alternate space instructions in ASI 0xD. Virtual address bits iu_dva[11:2] will be used to address the instruction cache in this mode; all other virtual address bits are ignored during these operations.

The internal memory data bus (mc_mdata) is used as input for fills on instruction cache misses, and as input for StA to ASI 0x0d.

6.0.3 Instruction Cache Tags

A instruction cache tag entry consists of several fields as follows.

Figure 6.2 - Instruction Cache Tag Entry



Field Definitions:

Reserved (Rsvd) - Bits [31:27,11.01] are not implemented, should be written as 0 and will be read as 0.

Physical Address Tag - This field contains the physical address of the data held in the cache line. The Instruction Cache Controller writes this field from bits [26:12] of the physical address (mm_ipa) of the line.

Valid - This bit indicates that the line contains data. This bit is set when a cache line is filled due to a successful cache miss; a cache fill which results in a memory parity error will leave the Valid bit unset. An alternate address space instruction cache flash clear operation will clear the valid bits of all of the instruction cache tag entries. A Flush instruction will clear the valid bit of the single line which is addressed by iu_dva[11:05] (regardless of the contents of that line).

There are two input sources to the instruction cache tag array The Physical Address bits needed for the tag are used for cache updates due to instruction cache misses. The internal memory instruction bus (mc_mdata) is used as input for alternate store operations.

System software can read and write the instruction cache tags by executing word-length LDA and STA (Load and Store Alternate)

instructions in ASI 0xC.; dva bits [11:5] will select one of the 128 tags; all other address bits are ignored.

6.0.4 Instruction Cache Fill

The memory block size of data fetched from memory on instruction cache misses is 32 bytes. Memory will always return 32 bytes of data, starting with the requested word first followed by the other word of the first doubleword and continuing with the three remaining doublewords (even word, then odd) which will wrap around a 32 byte boundary until the entire 32-byte block has been returned. The transfer rate is two words every three cycles from memory (two words of a doubleword, then a dead cycle). The Cache array is written during the cycle that each word appears on the mc_mdata bus. The following table illustrates the fill operation showing the order that words are written into the cache; 'D' represents a dead cycle in which no word is written:

Table 6.1 - Instruction Cache Fill Ordering

Requested Word	Order of fill
0	0, 1, D, 2, 3, D, 4, 5, D, 6, 7
1	1, 0, D, 2, 3, D, 4, 5, D, 6, 7
2	2, 3, D, 4, 5, D, 6, 7, D, 0, 1
3	3, 2, D, 4, 5, D, 6, 7, D, 0, 1
4	4, 5, D, 6, 7, D, 0, 1, D, 2, 3
5	5, 4, D, 6, 7, D, 0, 1, D, 2, 3
6	6, 7, D, 0, 1, D, 2, 3, D, 4, 5
7	7, 6, D, 0, 1, D, 2, 3, D, 4, 5

During an instruction cache fill, instructions from the missing line can be supplied to the IU or FPU by two separate mechanisms, these mechanisms are 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 can occur in all cycles of the fill sequence except the three dead cycles. The second form of streaming ("dead-cycle streaming") occurs only during the three dead cycles; any instruction word which has already been written into the RAM array can be accessed by reading the array. In a given cycle, the IU is only able to accept the instruction word which it is requesting; in some cycles, the IU may not be requesting any instruction at all, due to interlocks, multi-cycle instructions, or pipeline holds. If, in a given cycle, the IU is requesting a word which is available via streaming, then that word is supplied to the IU and the pipeline can advance.

6.0.5 Internal Memory Bus Interface

The instruction cache block interfaces to the internal memory bus (mc_mdata). Data for LdA from ASI 0x0d is driven onto mc_mdata by the Instruction Cache, under control of an enable signal from the MMU.

6.0.6 IU Instruction Bus Interface

The instruction cache block drives the IU instruction bus (ic_ibus). Instructions to the IU or FPU are sourced from either the mc_mdata bus (for bypass-streamed instructions on instruction cache misses, and for non-cached instruction fetches) or the instruction cache data array (for instruction cache hits, and for dead-cycle streamed instructions on instruction cache misses).

6.0.7 Instruction Cache Flushing

The instruction cache is implemented with a flash clear mechanism that is activated by any type of alternate store instruction to ASI 0x36. All instruction cache valid bits are reset (to zero) by this operation. Also, the FLUSH instruction always clears the single valid bit that is addressed by iu_dva[11:05], regardless of the contents of this tag entry.

6.0.8 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.

Accesses when the MMU is disabled and alternate cacheability is disabled (EN, AC bits of the MMU CR=0).

Accesses while the instruction cache is disabled (IE bit of the MMU CR=0).

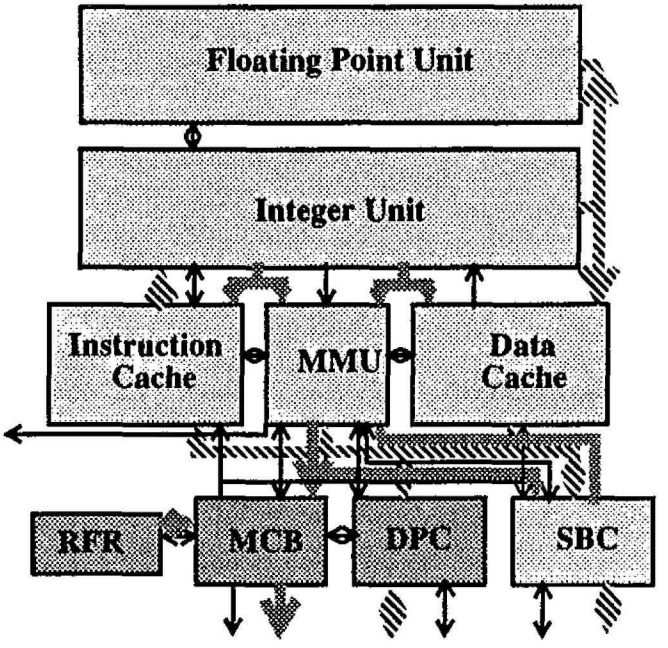
Accesses while in Boot Mode.

Accesses to sources in physical address spaces 1-7.

6.0.9 Diagnostic Strategy

Sublines and cache tags may be both read and written using ASI 0xD and 0xC respectively as previously discussed. The instruction cache will be structurally tested via the JTAG controller test ports. All register bits within the instruction cache and instruction cache tag are accessible via scan; on the chip level, all locations of these RAMs may be read or written by appropriate sequences of scan operations.

7.0Memory Interface



7.0.1 Overview

The microSPARC architecture allocates 256MB of space for the system memory (Physical address space'0', defined by mm_pa[30:28]), while the actual memory interface and the memory management unit can only support up to 128MB.

The following sections describe the general memory layout for a microSPARC-based system and then explains each of the logical blocks within the Memory Interface block.

The microSPARC 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).

7.0.2 Memory Subsystem

microSPARCMemory Interface is designed to primarily satisfy the basic system requirements, while providing sufficient capabilities to support future expansion.

The interface is designed with the following criteria in mind:

- 64 bit Data bus to increase memory bandwidth.
- 1 bit parity per word (32 bits) for reduced cost.
- Memory divided into blocks which can support different density devices This will allow relatively small memory increments with a small number of blocks.
- Allow for future higher memory requirements by supporting next generation of DRAM devices.

Typically a carefully laid out system board using the microSPARC chip would require 60ns DRAMs at 50MHz and 80ns DRAMs at 40MHz clock speeds. The designer however, should use the memory interface AC specifications in the microSPARC datasheet, to select the appropriate DRAM speed for a specific system and clock speed.

7.0.2.1 Memory Organization

microSPARCarchitecture defines a 28-bit physical address space for memory (PAS 0). This means a 256MB block for system DRAM. Electrically however, microSPARC uses only 27 bits of this address space, limiting the maximum memory for a microSPARC-based system to 128MB.

This 128MB is divided into 4 banks, each capable of addressing up to 32MB. The banks are defined as follows:

- Each bank is selected by a separate RAS line. There are a total of 4 RASes for DRAM banks (c_mc_ras_l[3:0]).
- The banks have a 64bit data path to microSPARC.
- All the banks use the same 2-bit CAS lines (c_mc_cas_l[1:0]), to select the upper or lower 32 bits (high or low word).
- All the banks use the same write signal (c_mc_mwe_l).
- All the banks use the same 22-bit multiplexed Row/Column address bus. At the time of finalizing the microSPARC memory interface, DRAM manufacturers were proposing 2 addressing schemes for 4Mx4 devices, an 11-row/11-column and a 12-row/10-column. MicroSPARC's memory interface will support DRAMs with an 11x11 matrix and DRAMs with a 12x10 matrix.

The memory interface is designed with the 4bit wide DRAM devices in mind. Using 16 such devices (or 2 SIMMs with eight devices on each) will provide the required 64bit wide data bus. In addition, each bank will require two 1bit wide devices of the same depth (If using SIMMs, one on each SIMM) to store the 2 parity bits.

Hence, each bank can be populated using one of the following configurations:

- 2MB (256Kx64) of data, using 16 of 256Kx4 devices for data and 2 of 256Kx1 for parity, or using 2 of 256Kx33 SIMMs.
- 8MB (1Mx64) of data, using 16 of 1Mx4 devices for data and 2 of 1MKx1 for parity, or using 2 of 1Mx33 SIMMs
- 32MB (4Mx64) of data, using 16 of 4Mx4 devices for data and 2 of 4Mx1 for parity, or using 2 of 4Mx33 SIMMs.

Note that a pair of double-density (e.g. 512Kx33 or 16Mx33) SIMMs will occupy 2 banks (Need 2 RASes).

7.0.2.2 Access to Unused or Unpopulated Memory regions

Any access to a location in the upper 128MB will be mirrored to its corresponding location in the lower 128MB and no errors will be generated.

Similarly, if a bank contains less than the defined maximum of 32MB, the real memory will be mirrored on the higher unused portions and an access from any of the unused sections will be mirrored to the corresponding location in the lowest block and no errors will be generated. For example, if a bank contains 2MB of real memory, this will be mirrored on the remaining 15 empty portions.

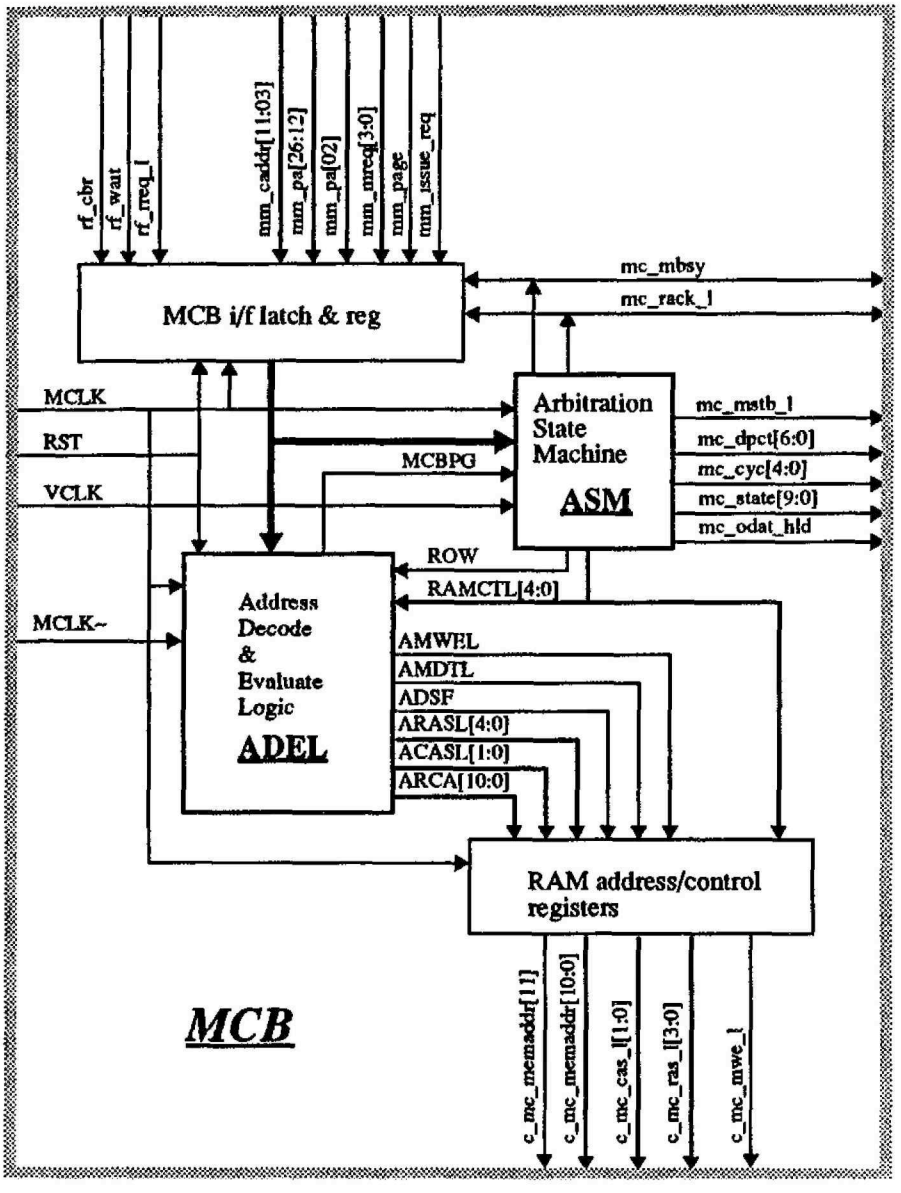
However, an access from a fully unused (empty) bank will complete, but it's result will be unknown and may cause a parity error.

7.0.3 Memory Control Block (MCB)

The operations that occur on the memory bus are data reads, writes, and read-modified-writes required for cpu execution, instruction fetches and prefetches, translation buffer accesses during table walks, reads and writes by IO devices, and all RAM refresh. The Memory Control Block (MCB) keeps track of the priorities of memory operations and completely controls the DRAM based main memory

As shown in the following diagram, MCB contains 2 major logic blocks, labeled "ASM" and "ADEL" which perform the memory arbitration and address mapping functions respectively This blocks will be described in the following subsections MCB also includes some input and output register blocks, which provide the synchronization among input and output signals.

Figure 7.1 - MCB block diagram.



7.0.3.1 Arbitration State Machine (ASM)

ASM is responsible for detecting the requests from MMU and Refresh blocks, arbitrate between them if necessary and grant the appropriate request. Once a request is granted, the MCB will carry out the requested memory operation which will consist of one or more memory cycles. The following table lists all the types of memory operations performed by the MCB, the possible request sources and the type and number of cycles involved.

Table 7.1 - 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 or do an SBus burst read of 16bytes.	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 and SBus 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 and SBus 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 and SBus 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 and SBus halfword 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.
d.rmw.b	MMU. Used for IU and SBus 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/VRAM	Will force a Cas-before-Ras refresh cycle to be performed on all DRAM and VRAM banks.

7.0.3.2 Arbitration for Memory Access and ASM Priority Scheme

ASM arbitration scheme is based on the following rules:

- All requests are checked at the end of each operation (for multi cycle operations, this means the end of last memory cycle) and:

If: no requests are pending, ASM will enter the idle state and will remain there until a request is detected.

If: only one request is pending, it will be granted and the operation will begin.

If: More than one request is pending, the one with the highest priority will be granted and the operation will begin The priorities are as follows:

g) MMU is the highest priority, except when the current cycle is also an MMU request, in which case it will be considered the lowest priority. This is to prevent bus locking as a result of back to back MMU requests.

h) RFR has the lowest priority, except when the current cycle is an MMU request, in which case it will have higher priority.

If: While in idle, an RFR request is detected, the state machine will advance to a "Check" state, where it'll look to see if an MMU request occurred just as RFR request was accepted If there are no MMU requests, ASM will continue to acknowledge the RFR request and do the cycle, else, it will do the MMU cycle.

7.0.3.3 Memory Operation Timing

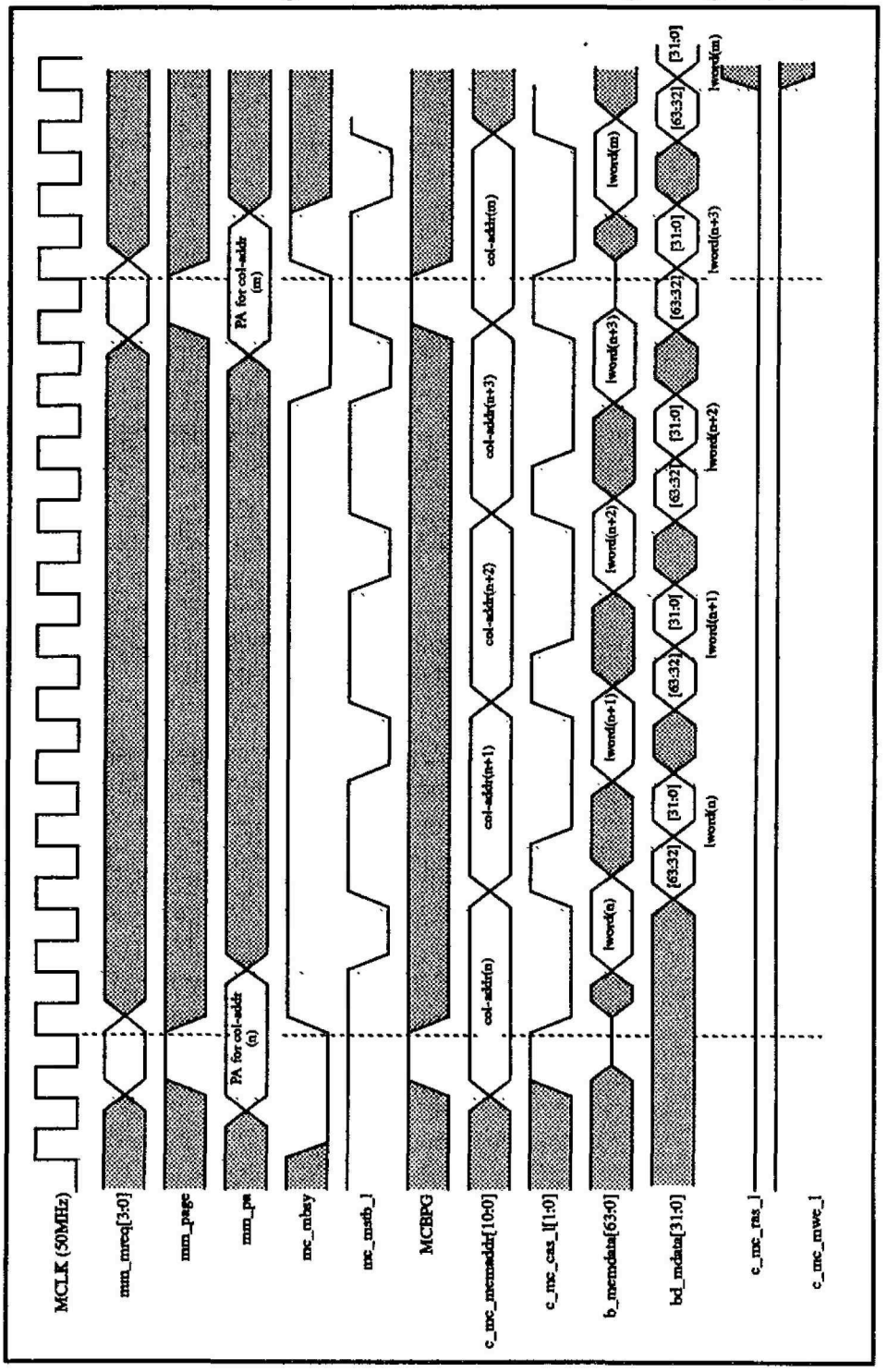
Following pages contain the waveform diagrams for some of the memory operations requested by MMU and carried out by the memory interface. Each operation-type is defined using the operation-name given in table-1 of this section.

The diagrams are functional and do not represent actual delays. Synchronous signals are clocked with the positive edge of the MCLK (derived from system clock, running at same frequency and assumed to have negligible skew) and are shown to be valid about half a clock period later. In case of the falling edge of the RAS signals only, the transition occurs after the negative-edge of the system-clock

In addition, the mm_mreq[3:0] is shown valid for 1 clock periods. This indicates the clock-cycle during which MMU asserts the mm_issue_req signal

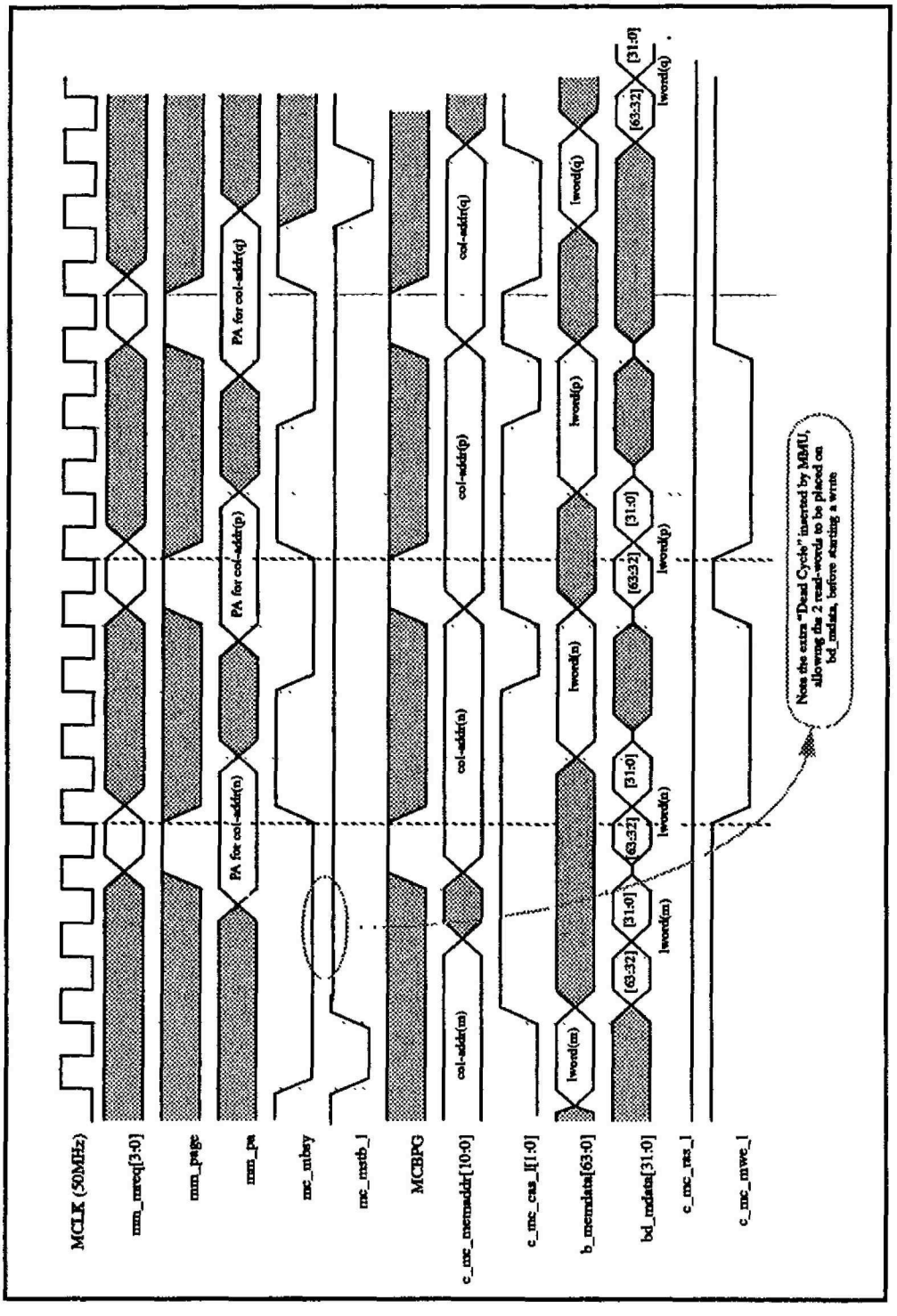
The waveforms are provided only as a general reference and do not reflect details such as the word/hword/byte order relative to the address or the MMU request type etc.

Figure 7.2 - MMU I-fetch beginning in page-mode



MMU I-fetch beginning in page-mode, followed by another MMU read request from same DRAM-page.

Figure 7.3 - MMU page-mode write after a read



MMU page-mode write after a read, followed by another write, followed by another read, all from same page.

Figure 7.4 - Non-paged write cycle, shown following a read

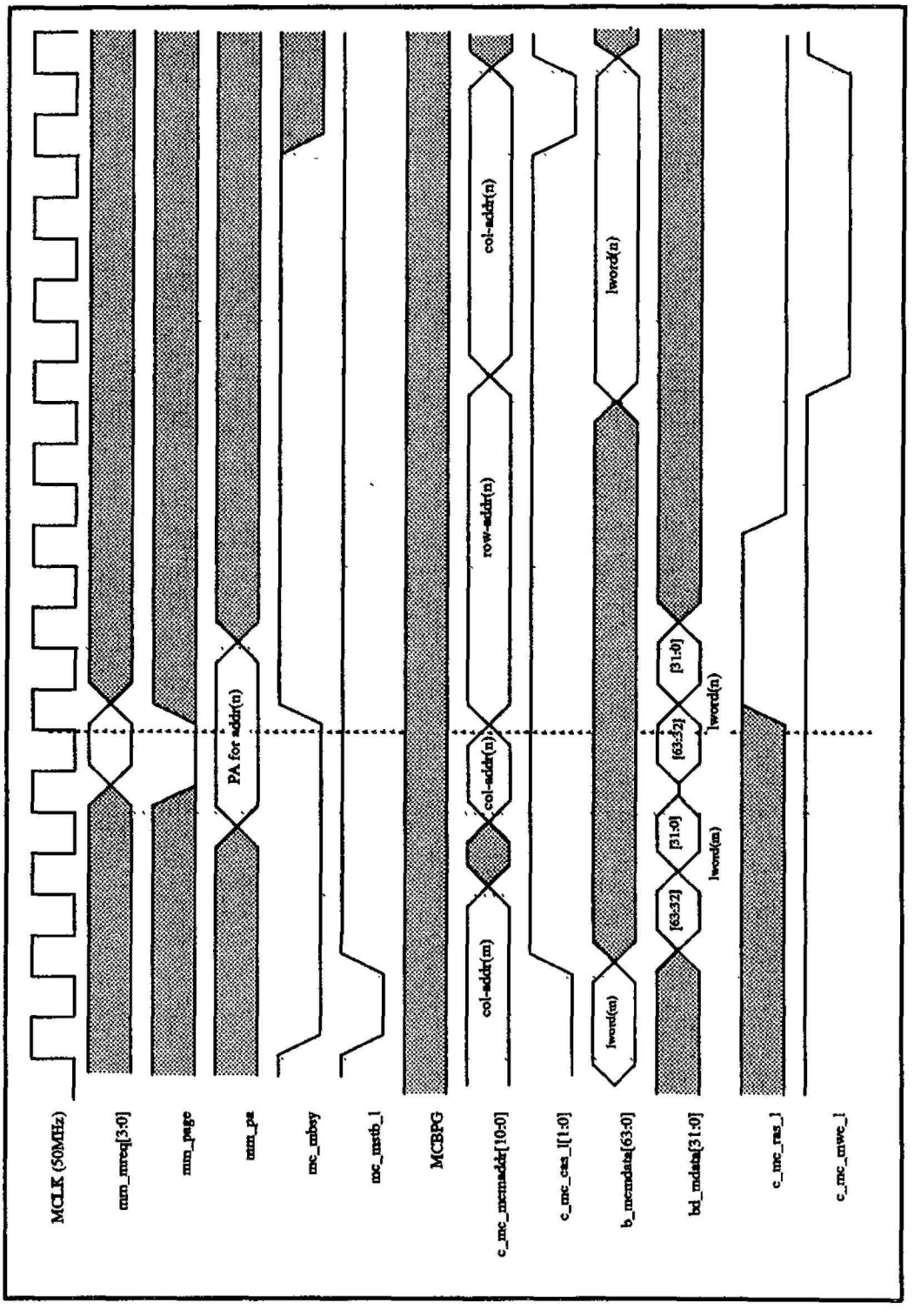


Figure 7.5 - Non-paged read cycle, shown following a read

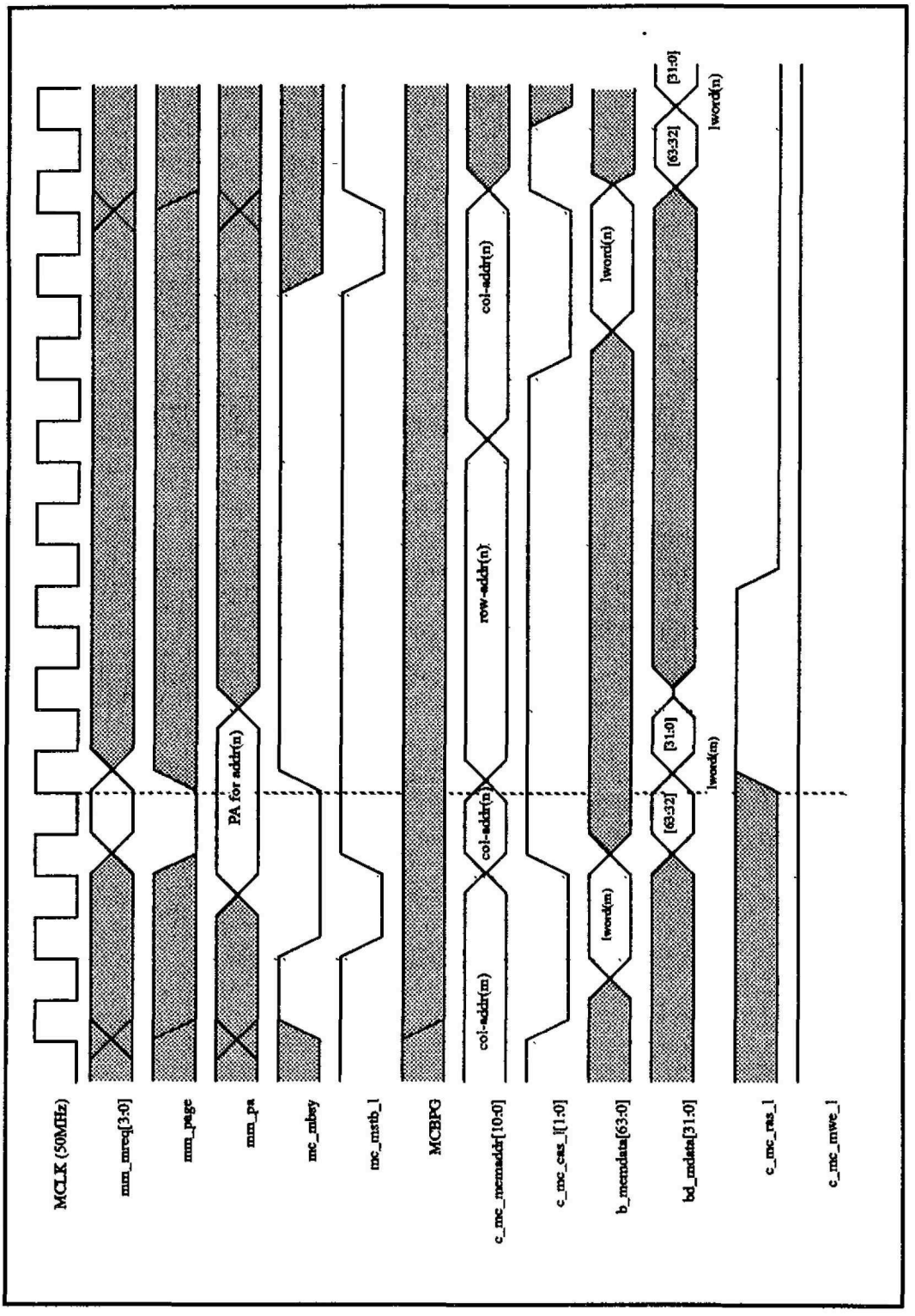
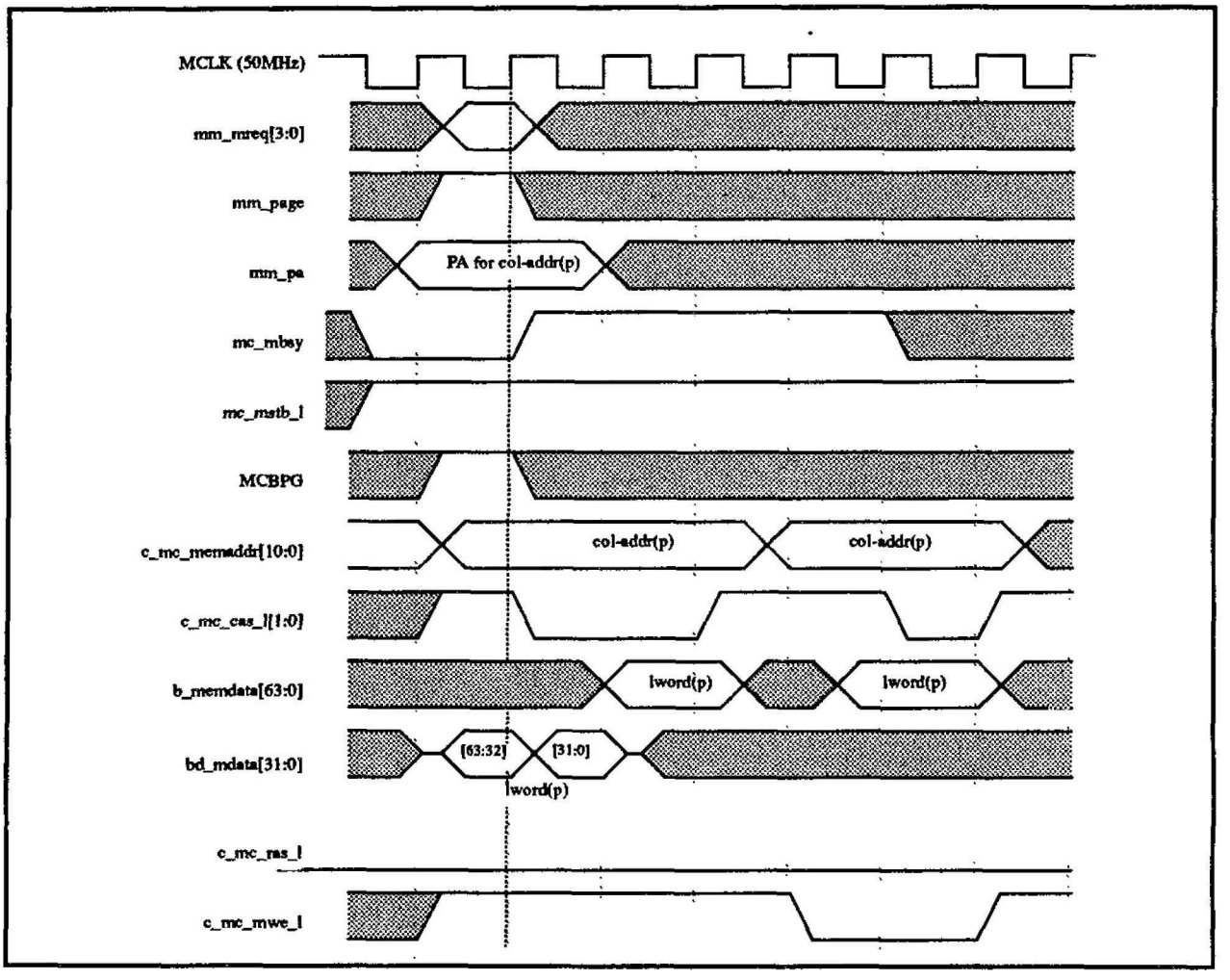
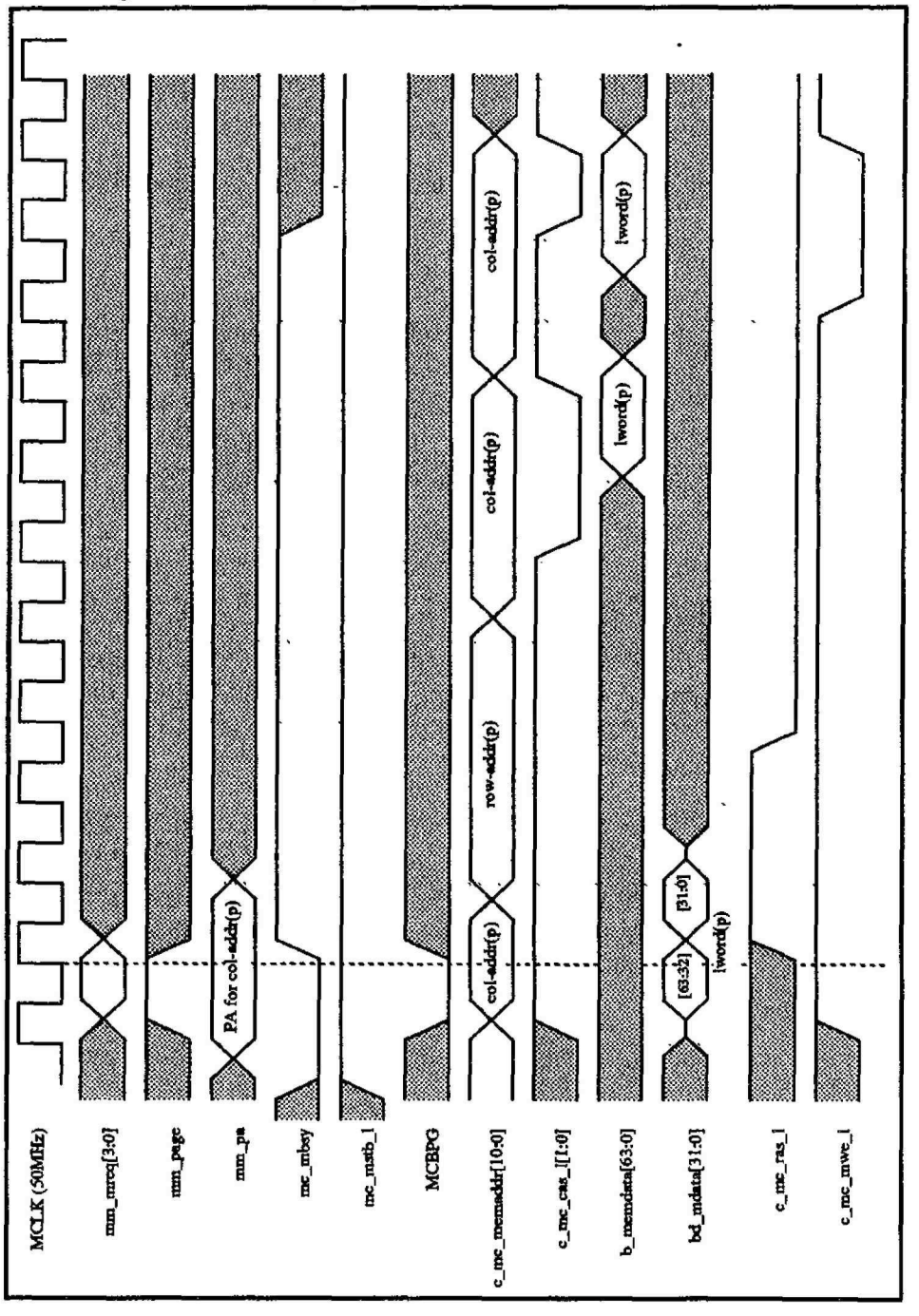


Figure 7.6 - Paged Byte/Halfword (8/16 bit) write cycle.



Paged Byte/Halfword (8/16 bit) write cycle, generating a hardware controlled Read-Modify-Write sequence.

Figure 7.7 - Non-paged Byte/Halfword (8/16 bit) write cycle.



Non-paged Byte/Halfword (8/16 bit) write cycle, generating a hardware controlled Read-Modify-Write sequence.

7.0.3.4 Address Decode & Evaluate Logic (ADEL)

This block primarily monitors the address and function-select signals coming from 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 row/column address and memory control signals required for the current operation out to memory.

The mapping of system memory is discussed in the following section.

7.0.3.5 Address Mapping For System DRAM

From the 31 bits of the physical address bus driven by MMU block (mm_pa[30:0]), the three MSBs (mm_pa[30:28]) represent 1 of the 8 physical address spaces (PAS) as defined in microSPARC architecture. From these, only PAS0 is of concern to MCB, since an MMU request from MCB will only be 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, ADEL uses the mm_pa[26:02] address bits to complete its decode. The following table describes the decode scheme used for system memory.

A maximum of 512 memory cycles can be made from a contiguous block, while remaining within a DRAM page. This gives a maximum of 4K (512x64) block size which can theoretically be accessed using page mode cycles only.

A point to note from the table below, are the staggered decoding of mm_pa[24:21] for c_mc_memaddr[10:9]. This was necessary in order to allow different size devices (256Kx4, 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 the table, mm_pa[23] is used as both c_mc_memaddr[10] for column address and c_mc_memaddr[11] for

row address. This is to cater for the 2 different 4Mx4 DRAM architectures, 11x11 matrix and 12x10 matrix.

Table 7.2 - Physical Address decode for System Memory

PA	Decode
30-27	Not Used System memory limit is 128MB
26-25	Decode to select 1 of 4 RASes: 00 RASL0 1st 32MB bank 01 RASL1 2nd 32MB bank 10 RASL2 3rd 32MB bank. 11 RASL3 4th 32MB bank.
24	Decoded as row address bit 10 (c_mc_memaddr10). Required for 16MBit DRAMs.
23	Decoded as column address bit 10 (c_mc_memaddr10) and row address bit 11 (c_mc_memaddr11) Required for 16MBit DRAMs See text for more information
22	Decoded as row address bit 9 (c_mc_memaddr9) Required for 4MBit DRAMs
21	Decoded as column address bit 9 (c_mc_memaddr9) Required for 4MBit DRAMs and up
20-12	Decoded as row address bits 8 to 0 (c_mc_memaddr[8:0]). Required for 1MBit DRAMs and up.
11-3	Decoded as column address bits 8 to 0 (c_mc_memaddr[8:0]). Required for 1MBit DRAMs and up
2	Decoded to select one of 2 CASes: 0 CASL0 Lower data word (bd_mdata[31:0]) 1 CASL1 Higher data word (bd_mdata[63:32])
1-0	Not used for external decode Byte and halfword writes are achieved by MCB and DPC doing a read, update, write sequence. This bits are used then, to select the appropriate data fields.

7.0.4 Data aligner and Parity Check/ generate logic (DPC)

DPC is responsible for transferring data between external memory data bus and the internal data path as well as generating and checking of parity for system main memory (DRAM).

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

DPC also contains the parity generation and checking logic. The parity is composed of 1 bit per word (32 bits) and is used for system DRAM only.

Type of parity operation for the system DRAM is determined by the state of the Parity Control bit (PC) in the Processor Control Register as described in the following table:

Table 7.3 - Parity Control Definition

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

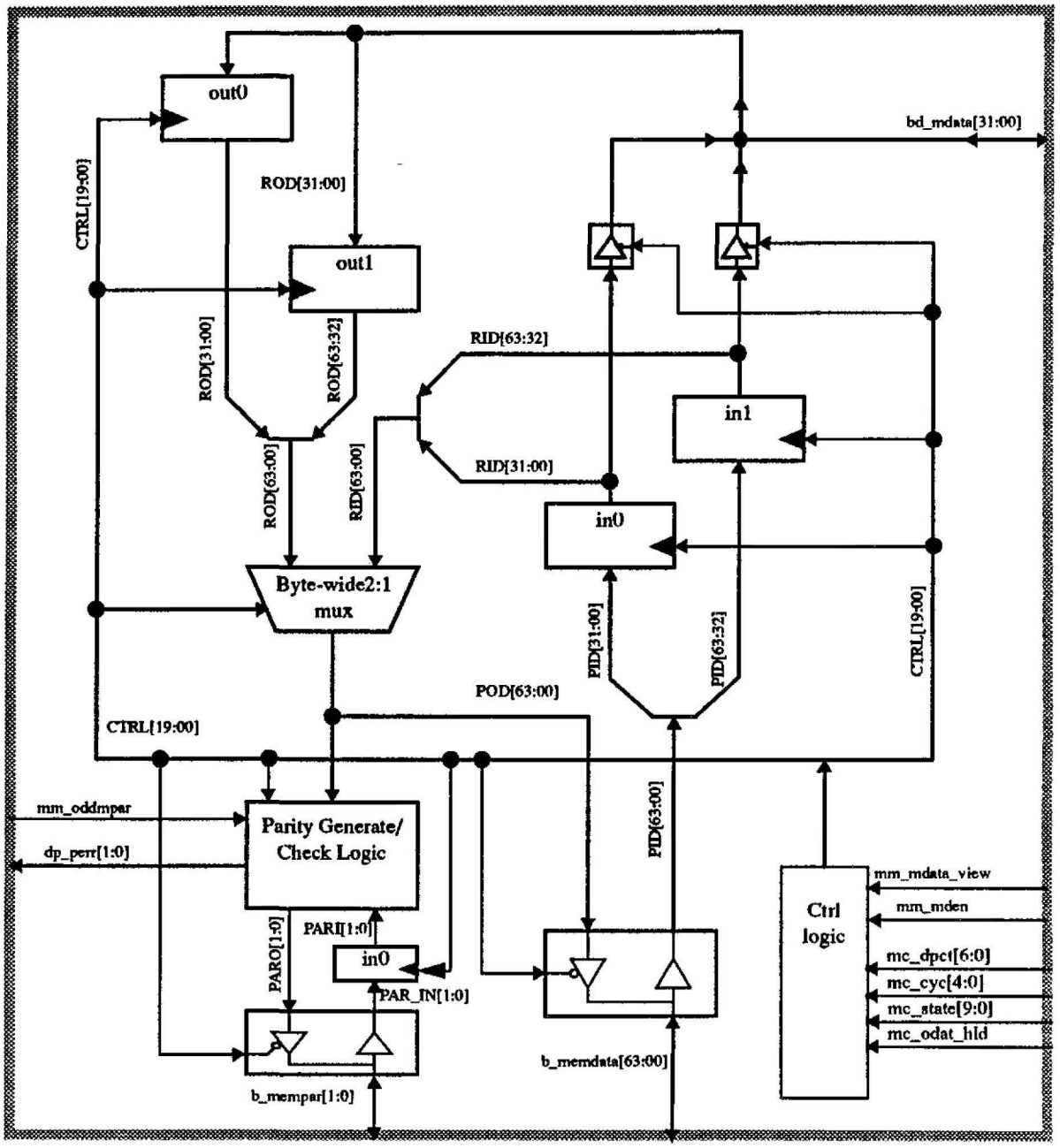
Since system parity is 1-bit per word, any byte or halfword store operation, will result in a hardware controlled read-modify-write cycle. During the read part of such operation, the word parity will be checked and if an error is detected, a parity error will be generated. After the word has been updated to contain the new byte/halfword, a write operation will be performed, which will also update the parity.

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

DPC block diagram, given below, shows the basic data paths connecting the 64bit external memory bus (b_memdata[63:00]) to the 32bit internal one (bd_mdata[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 mux.

The alignment mux is also used to combine and produce the output data during a read modify write sequence. The complexity of this mux 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 c_dp_mdata (D or I cache, IU, SBus controller).

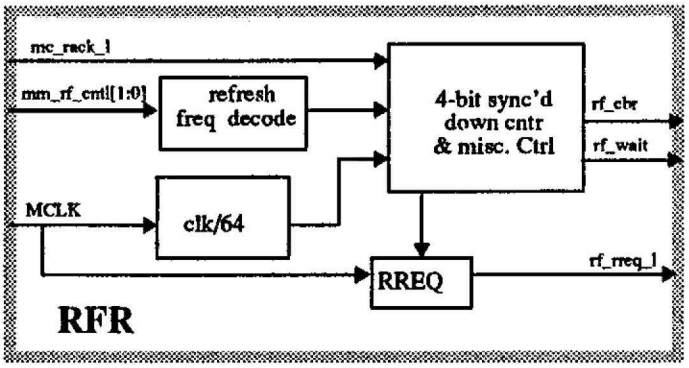
Figure 7.8 - Datapath and Parity Control (DPC) block diagram



7.0.5 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.

Figure 7.9 - RAM Refresh Control block diagram.



RFR refresh rate can be selected by programming 2 bits of the Processor Control Register according to the following table. These bits are then passed to RFR as mm_rf_cntl[1:0] input bits, which controls the rf_rreq_l rate.

Table 7.4 - Refresh Rate Control bits.

mm_rf_cntl [1:0]	Refresh interval
0 0	Assert a refresh request once every 128 MCLK periods. With this setting, adequate refresh is guaranteed for MCLK values of down to 8.6MHz This is the default after power up.
0 1	No Refresh!
1 0	Assert a refresh request once every 512 MCLK periods. With this setting, adequate refresh is guaranteed for MCLK values of down to 35MHz.
1 1	Assert a refresh request once every 768 MCLK periods With this setting, adequate refresh is guaranteed for MCLK values of down to 52MHz.

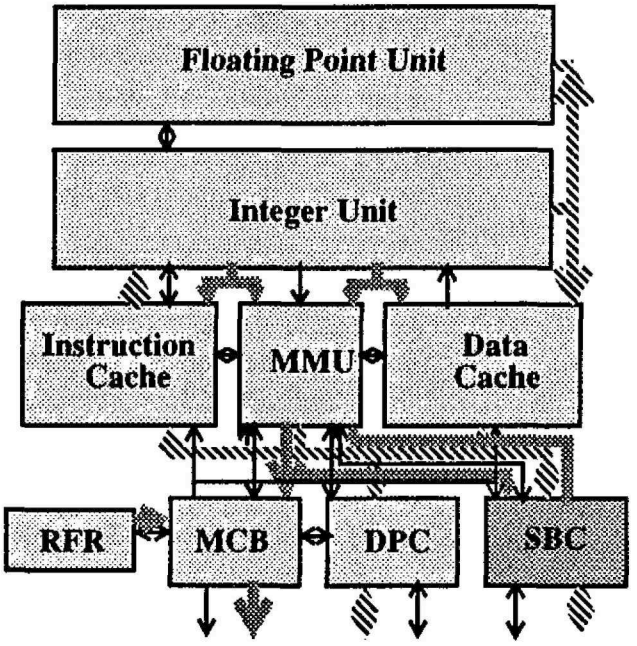
The RFR is also responsible for initializing the DRAMs on power-up.

After power-up and before they can be reliably used, DRAMs require a 200us "Wait" period followed by 8 Cas-before-Ras refresh cycles.

For systems built around microSPARC, the reset must remain active for at least 200us after power-up, to satisfy the "Wait" period. In systems using the NCR 89C105 chip, the reset is supplied by the NCR 89C105 chip. On power-up the NCR 89C105 chip guarantees an active reset duration of ~200ms and for subsequent software initiated resets it will force reset active for ~1024 SBus clocks (~50us).

After an active reset, the "mm_rf_cntl" bits which reside in the MMU's PCR register are set to "00" (See table 2.7.4), 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 will perform 8 CbR refresh cycles, completing the DRAM initialization cycle. After that, RFR will negate both "rf_cbr" and "rf_rreq_l" signals, allowing MCB to proceed to it's normal operation state.

8.0 SBus Controller



8.0.1 Overview

The SBus Controller (SBC) refers to the I/O subsystem that handles input and output between local resources, including the CPU, system memory, and control space, and all external system resources. The SBC is implemented as an SBus in accordance with the SBus Specification Rev. A.2. The SBC supports:

- Programmed Input/Output (PIO) transactions between the CPU and SBus slave devices
- Direct Virtual Memory Access (DVMA) transactions between SBus masters and local resources. (referred to as local DVMA)
- Direct Virtual Memory Access (DVMA) transactions between SBus masters and other SBus slave devices. (referred to as bypass DVMA)

Standard SBus features, such as dynamic bus sizing, reruns, atomic transactions, bus arbitration, burst transfers (up to 16 bytes), watchdog timer, and error reporting are fully supported. Interrupts and SBus Reset are not implemented in the SBC; these functions are handled elsewhere

The SBC plays many SBus roles. It serves as an SBus controller by arbitrating bus requests, translating virtual addresses, enabling slave cycles, etc. In addition, the SBC may act as either an SBus master or an

SBus slave. For PIO transactions, the SBC acts as an SBus master. For DVMA transactions, the SBC can act as either a slave or have no role at all, depending on the target of the DVMA transaction as indicated by the physical address. For local DVMA, the SBC has a role as both a bus controller and a slave device. For bypass DVMA, the SBC has a role as a bus controller only, not as a slave.

PIO transactions consist of an SBus slave cycle only; the address translation is done in advance of the bus acquisition.

PIO transactions occur when the CPU executes loads or stores to I/O (SBus) space. In the case of a PIO write transaction, the write is posted. Processing in the CPU continues while the SBus transaction completes in the SBC. A stall will occur only if another PIO transaction is attempted before the previous PIO write transaction completes. In the case of a PIO read transaction, processing is always stalled until the data becomes valid at the end of the SBus transaction.

DVMA transactions occur when an SBus master has acquired the bus in order to execute a transaction to a slave. A DVMA transaction consists of an address translation cycle and a slave cycle. The target of the slave cycle is determined once the translation cycle completes. The slave target can be either a local resource, defined as locations in either system memory or system control space, or another SBus device.

During the address translation cycle, the SBC obtains a virtual address from the DVMA master and is submitted to the MMU for translation. The MMU returns a physical address. The type of DVMA slave cycle, either local or bypass, is determined from the physical address.

A significant distinction concerning memory data transfers is that since system memory is a local resource, it is necessary for memory data to pass through the SBC; "fly-by" memory data transfers are inappropriate. Local DVMA slave cycles have two distinct, sequential operations in the SBC: a data get followed by a data put operation. A data get operation loads up to 16 bytes of data into an internal data store. A data put operation transfers the data from the internal data store to a destination. The data get operation can either be an internal data transfer

or an SBus cycle, depending on the read/write direction; the data put operation will be the other.

Figure 8.1 - Data Get and Data Put

	Data Get	Data Put
Read	Internal Data Transfer	Sbus Cycle
Write	Sbus Cycle	Internal Data Transfer

For local DVMA slave read cycles, an internal data transfer occurs during the data get stage, and an SBus slave cycle occurs during the data put stage. In this case, the data get operation shows up as a pause between the SBus translation cycle and the SBus slave cycle.

For local DVMA slave write cycles, an SBus slave cycle is during the the data get stage, and an internal data transfer is during the data put stage. In this case, the DVMA transaction is finished after the slave cycle completes in the data get stage. The current cycle is not held up during the internal data transfer, but data put stage may show up as bus latency before the next translation cycle occurs.

Bypass DVMA slave cycles do not involve the SBC as a slave target. The data transfer is between an SBus master and another SBus slave. There is no data get and data put operations in this case.

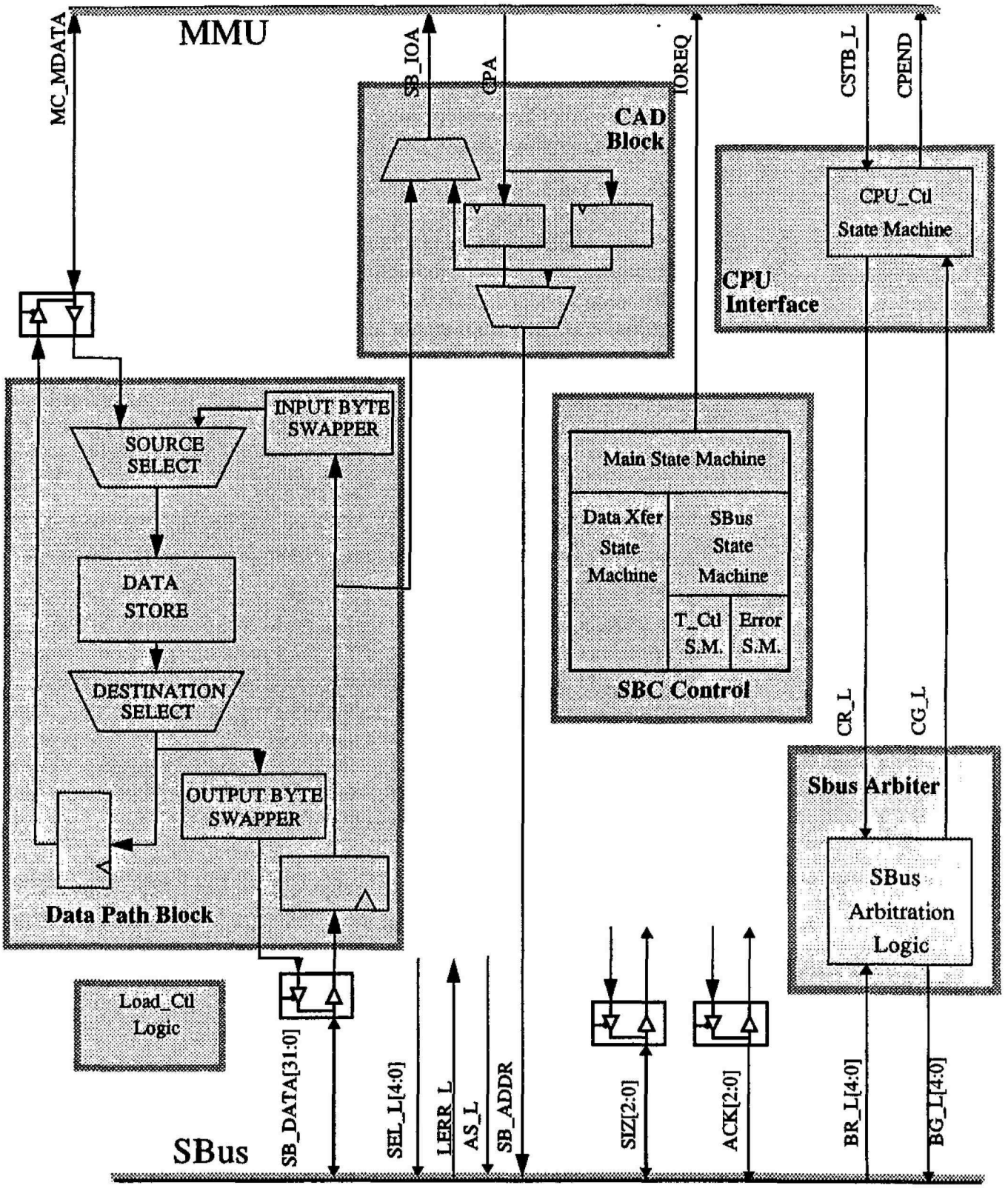
As a bus controller, the SBC has to handle bus errors and watchdog timeouts. Bus errors that occur during PIO cycles are handled by making the current state of the bus cycle available to the MMU. Bus errors that occur during DVMA cause the SBC to intercept the slave cycle from the intended slave target and, itself become the slave target in order to terminate the cycle with an error. Watchdog timeouts occur when an internal timer expires and the SBC terminates the slave cycle with an error.

The subcomponents of the SBC are the CPU Interface, Address Steering, SBus Arbiter, Main Control, Data Transfer Control, SBus Slave and Target Control, Data Path and Control, and Error Control blocks. These subcomponents are schematically shown in the following block diagram. A further description of these blocks is given in the following sections.

If it is appropriate, a state diagram is provided, along with a narrative walk-through. The state diagrams are provided for heuristic purposes. The intent is to purposely omit descriptions of some logic that tends to

cloud the understanding of the general functionality. This logic, used for such implementation-specific purposes as logic synthesis and timing aids, would detract from the overall comprehensibility of the SBC block.

Figure 8.2 - SBus Controller Block Diagram



8.0.2 CPU Interface

The CPU interface block handles the MMU/SBC handshake protocol, arbitrates for the SBus, catabolizes double word PIO into single word PIO, if appropriate, and supports dynamic bus sizing and bus cycle reruns. The data sizes supported for PIO cycles are byte, half byte, word and double word. There is also a high-performance feature that allows for very fast PIO writes to occur, which is especially important for certain operations that require fast output, such as graphics.

The CPU interface is double buffered, meaning that a copy of the entire state of the current cycle is retained for both PIO reads and PIO writes. The double buffering is necessary in the event of dynamic bus sizing, catabolic double word transactions or bus cycle reruns. The buffering also permits a DMA address translation to occur concurrent with a PIO transaction. This is important in deadlock avoidance.

The deadlock could occur when simultaneous PIO and DVMA transactions occur. The deadlock is avoided by buffering the entire state of the PIO transaction, and allow the DVMA transaction to proceed. Upon completion of the DVMA transaction, the PIO transaction, which had been retained in the SBC, would proceed.

The PIO buffers effectively provide a single element of a write buffer, since the CPU continues to execute instructions without waiting for a PIO write to complete.

A walk-through of the CPU State Machine (CSM) is given below. A set of signals, CSTB_L, CPEND, and IOREQ form a handshake between the MMU and the SBC. A PIO transaction is issued from the CPU through the MMU to the SBC by the assertion of CSTB_L. This occurs only if the SBC is not busy, as indicated by CPEND. De-assertion of CPEND indicates that the SBC is not busy and free to receive a PIO cycle. In the case of PIO reads, IOREQ is used to signal that data is ready. (IOREQ is also used for other various MMU/SBC communications).

The CSM remains in idle until CSTB_L is received. For a simple PIO transaction, control transitions into a bus request state where it remains until the bus is acquired. Once the bus is acquired, the state changes, and holds until the bus is relinquished. Next, in the case of PIO reads, control passes to a housekeeping state before returning to idle; in the case of PIO writes, control returns directly to idle.

Special states in CSM support catabolic double word transactions. Whenever a double word PIO transaction is attempted to an SBus device that does not support bursts, the SBC automatically catabolizes the double word burst transaction into single word transactions. (Status bits

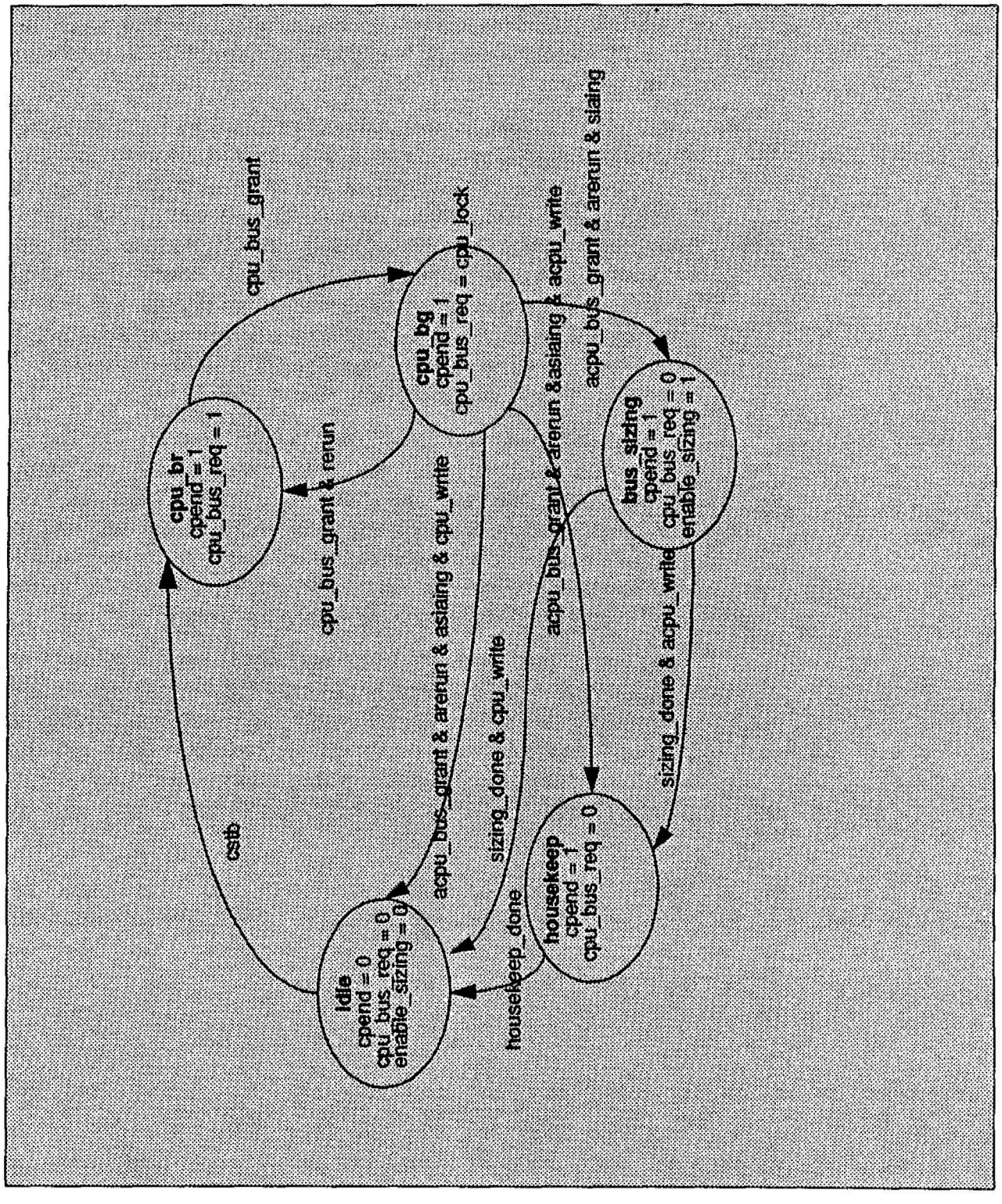
from the MMU Slot Configuration Registers indicate SBus device burst-handling capabilities) The MMU is held off during this time by CPEND. This operation is transparent to the MMU. Dynamic bus sizing and bus cycle reruns can occur in either portion of the catabolized transaction.

While a PIO cycle is in progress, as indicated by the CSM grant state, a dynamic bus sizing operation may occur which would cause control to branch to a special holding state. Simultaneously, the dynamic bus sizing state machine transitions from idle to handle this operation. When finished the CSM is signaled and control continues as if a simple PIO transaction had occurred. To improve latency during dynamic bus sizing, an attempt is made to keep the follow-on cycles atomic. If a rerun occurs, however, other DVMA masters are given a higher bus arbitration priority and the atomicity of the follow-on cycles will be broken. Reruns are supported whenever they occur.

Reruns can occur during any phase of a PIO transaction; during simple PIO transactions, dynamic bus sizing, atomic cycles, or catabolized transactions. When a rerun occurs, the transaction is ended, the bus is relinquished, and the cycle begins anew. Provisions are made the bus arbitor to allow any requesting DVMA masters onto the bus, before the PIO cycle is retried. For this reason atomic transfers may not work properly when the SBus slave recipient is capable of reruns.

A special speed path is built into the CSM to allow fast PIO writes. A prerequisite for this operation is that the data size must be a word (or double word) and must not be atomic. Another necessary condition is that the SBus slave device must respond with word acknowledge. If this criteria is met, then PIO word writes can sustain a bandwidth of 33 Mbytes/sec at 25 MHz. (PIO double word writes can sustain 50 MBytes/sec.).

Figure 8.3 - CPU State Machine



8.0.3 Address Steering

The Address Steering Block handles the address generation function for SBus transactions and local resources data transfers. The block insures that the proper SBus physical address is valid and stable whenever address strobe is asserted In addition, this block generates the address used during a request for local resources.

There are two sources for an SBus physical address; the CPU generates a virtual address during PIO transactions and the SBus master generates a virtual address during DVMA transactions. In both cases the MMU translates the virtual address to a physical address. Since PIO and DVMA transactions can overlap, both physical addresses must be retained by the Address Steering Block. The only time the CAD block manipulates the SBus physical address is during double word catabolism and dynamic bus sizing.

In order to deal with such implementation-specific processes as memory burst order and local resource transfer sizes, the CAD block manipulates some low-order address bits to simplify data transfer control. In either case, data is transferred properly and control logic is simplified.

8.0.4 SBus Arbiter

The SBus Arbiter handles bus requests from the CPU and as many as five DVMA masters. The SBus Arbitor employs all fairness, and arbitration protocol as outlined in the SBus Specification A.2.

The fairness algorithm utilizes a token, which is passed round robin style. All six masters are given tokens which are prioritized based on the last master to have owned the bus. The requesting master with the highest priority is granted the bus. Once that master is finished with the bus, new tokens are assigned. The last owner is given the lowest priority.

The CPU is treated as one of the six masters. In this regard, the CPU master is indistinguishable from any other DVMA master. In addition to this, there are two ways in which the CPU is given special treatment. If the bus is free and is not about to be granted, the CPU has the ability to anticipate that its request will be granted. In this fast-bus-access case, the CPU will forego waiting for the bus grant in order to begin the bus cycle.

Another special case is made during times when a PIO transaction is dynamic bus sized. An attempt is made to keep the follow-on cycles atomic with the first cycle (although a rerun will cause the atomicity to be broken) in order to help the latency of that cycle.

A walk-through of the Arbitor State Machine (ASM) is given below: Control begins in Idle where it remains until a sampled version of at

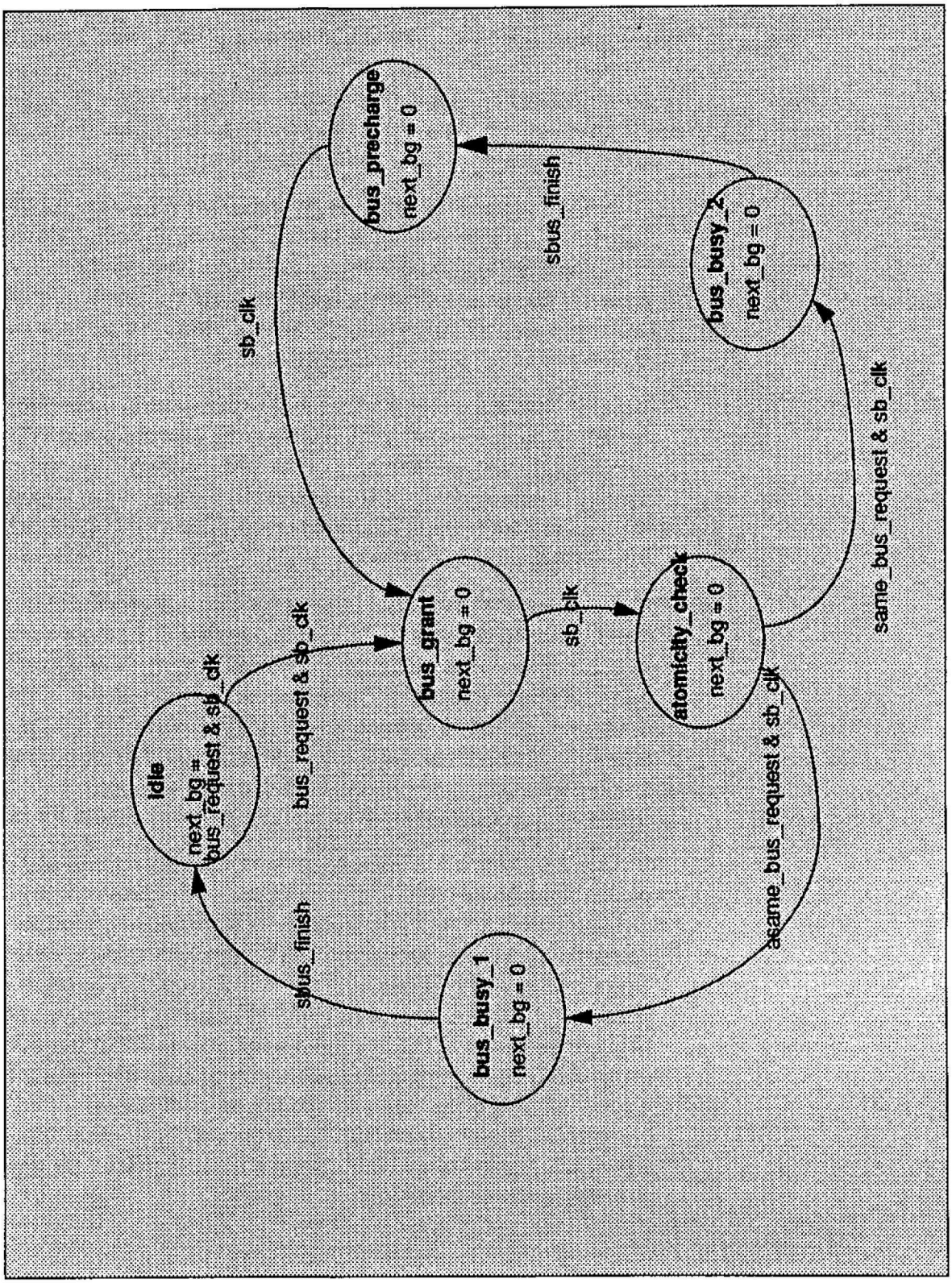
least one bus request is detected. Control transitions into the Bus_Grant state at the same time that the requesting master with the highest priority token is granted the bus.

On the proper phase of SBus clock, control moves into the Atomicity_Check state, where the current bus owner's request line is sampled for atomicity. For DVMA masters, the virtual address is latched during Atomicity_Check and a translation request is issued If the request is still active, control branches to a special atomicity loop; otherwise control passes into a Bus_Busy state. Here it remains until the bus cycle is finished and then returns to Idle.

If the atomicity loop was taken, a different Bus_Busy state is entered. This Bus_Busy state is very similar to the first, except instead of entering Idle upon completion a Bus_Precharge state is entered. On the proper phase of SBus clock, control transitions into the Bus_Grant state. Other masters, however are not given an opportunity to compete for the bus and another bus cycle is granted to the previous master.

Once in the Bus_Grant state, control cannot distinguish how it arrived in that state (from either the Idle or the Bus_Precharge state). This means that each cycle is separate onto itself, and the atomicity check is made once during each bus cycle. It is possible for a master to retain the bus by constantly requesting it. (although good SBus citizens would never do this).

Figure 8.4 - Arbitor State Machine



8.0.5 Main Control

The Main State Machine (MSM) controls the internal data store, issues data transfer and translation requests to the MMU, and generally acts to coordinate the other state machines in the SBC. One of the major functions of the SBC is to move data between local resources and SBus devices. The SBC has to fill the internal data store by a get operation and then to empty the data store with a put operation. The MSM controls the above-mentioned get and put operations.

A walk-through of the MSM is given below: When in the Idle state, MSM monitors the state of ASM through two signals, CG, which indicates that the CPU has been granted the bus and XLAT, which indicates that a valid virtual address has been received and is ready to be translated. Control remains in the Idle state until one of these two signals is detected.

If CG is detected and it is a PIO write, then control transitions to the SputW state. In the case of PIO writes, the data store was filled concurrently with the issuance of the PIO cycle. All that remains to be done is to put the data to SBus space. An SBus cycle is issued. Control remains in SputW for the duration of the SBus cycle and then returns to Idle upon completion.

If CG is detected and it is a PIO read, then control transitions to the SgetR state. In the case of PIO reads, the data store must be first filled by a get operation from SBus space. An SBus cycle is issued from SgetR. Control remains in SgetR for the duration of the SBus cycle Under certain conditions, such as reruns, dynamic bus sizing and catabolic double word cycles, control returns to Idle in anticipation of another CG. Upon completion, control passes to DputR. In DputR valid data is indicated by assertion of IOREQ. The data is then put to the CPU.

If XLAT is detected, a DVMA transaction is in progress and control passes to the Xlate state. A translation cycle is requested from the MMU through IOREQ. PA_VAL, from the MMU, indicates that the translation cycle has completed and the physical address is available Upon receipt of PA_VAL, a 3-way branch occurs. The target of the DVMA cycle is determined from the physical address; it is not possible to know the target of the DVMA transaction until the transaction is complete. If the target is not system memory or control space, then Sbyp state is entered. If the target is memory or control space, then control branches to DgetR for DVMA reads; SgetW for DVMA writes. If any error had occurred at any time during the translation cycle, a special 4th branch, the Pet state, is entered.

From the Sbyp state, the only function of the SBC is to act as an SBus controller; the SBC has neither a master or a slave role. An SBus cycle is simply issued and upon completion of the cycle control returns to Idle.

From the DgetR state, a get from memory or control space is required in accordance with a DVMA read transaction. Implicit to the read translation request is a data request if the target is determined to be system memory or control space; a separate request for data is not required Once the data store is filled, control moves to Sputr. If a parity error occurs, the Pet state is entered.

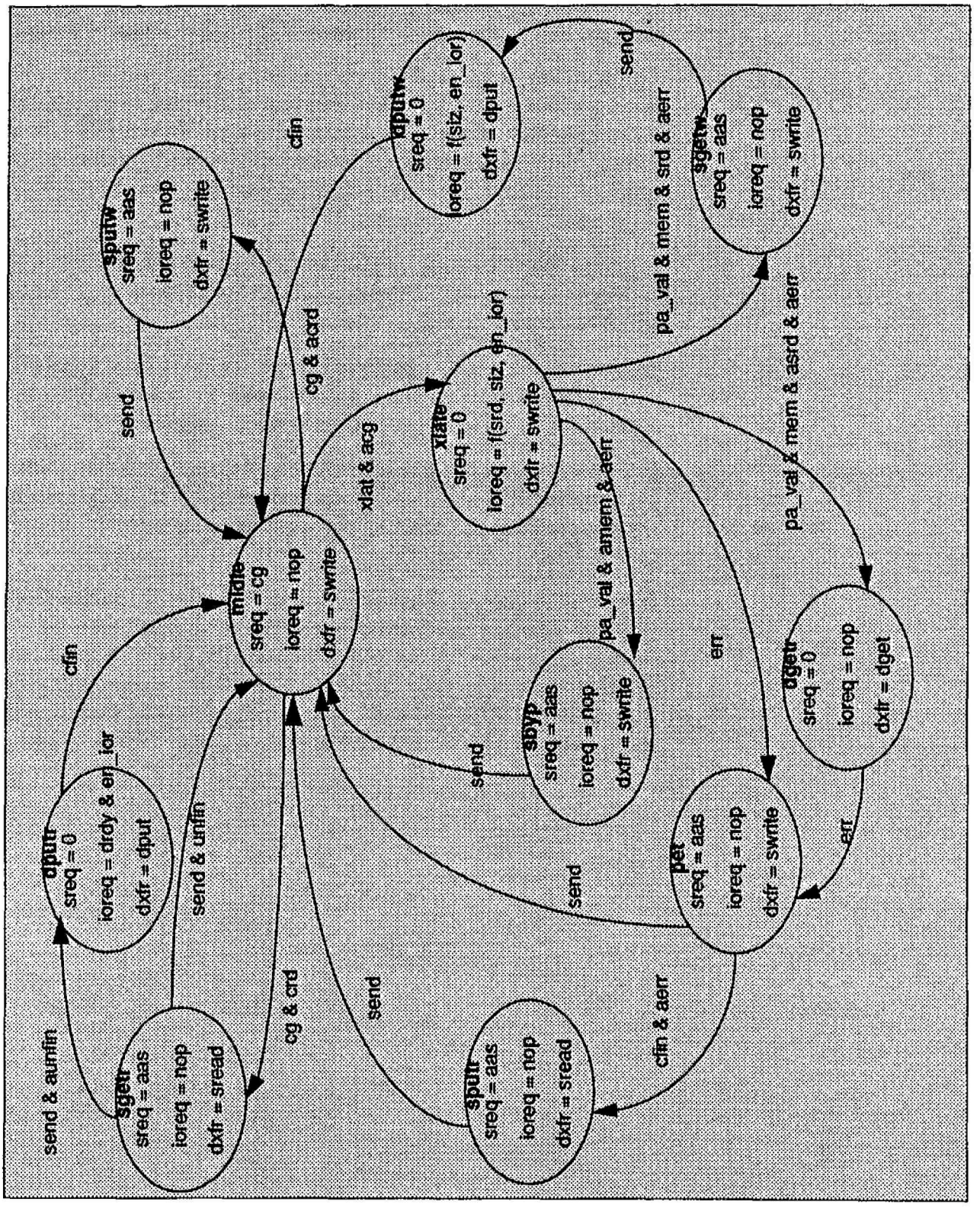
From the SputR state, the DVMA read transaction is completed by issuing an SBus cycle. Control remains in SputR state until completion and then transitions back to Idle.

From the SgetW state, a get from SBus space is required in accordance with a DVMA write transaction. An SBus cycle is issued, the data store is filled, and control moves to DputW upon completion.

From the DputW state, the DVMA write transaction is completed by a put of the data to system memory or control space. IOREQ is asserted to request a write cycle. After the put operation is complete control returns back to Idle.

From the Pet state, an SBus cycle is issued, but the SBus controller must intervene since an error has occurred. The slave select must be suppressed in order for the SBC to become the target. A special signal is sent to the Target State Machine which has the responsibility of driving SB_ACK[2:0] to indicate an error in this case. Control remains in Pet until completion of the SBus cycle and then returns to Idle.

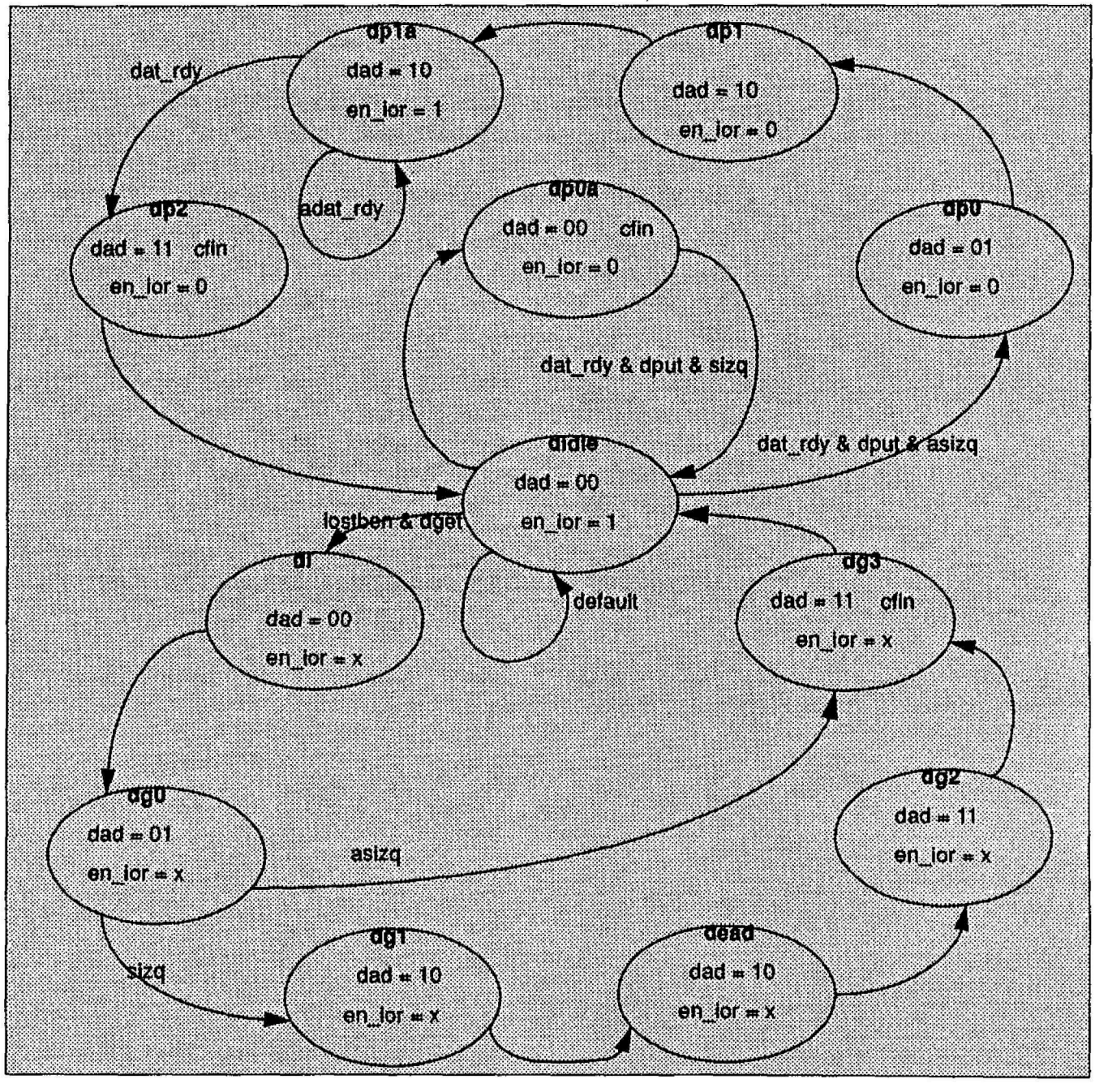
Figure 8.5 - Main State Machine



8.0.6 Data Transfer

The data control state machine (DSM) controls the movement of data between system memory or control space and the internal data store. The DSM monitors transaction size information and data transfer signals from the MMU. The data is counted and a signal, CFIN, is asserted upon completion.

Figure 8.6 - D_ctl State Machine



8.0.7 Slave Control Cycle

The SBus control state machine (SSM) is charged with tracking the progress of the current SBus cycle by monitoring the Transfer Acknowledgment (ACK) and terminating the cycle once completed or upon an error detection. The SSM does not differentiate between the ACKs from the TSM and other external ACKs; it treats the TSM as any other slave capable of responding with an ACK.

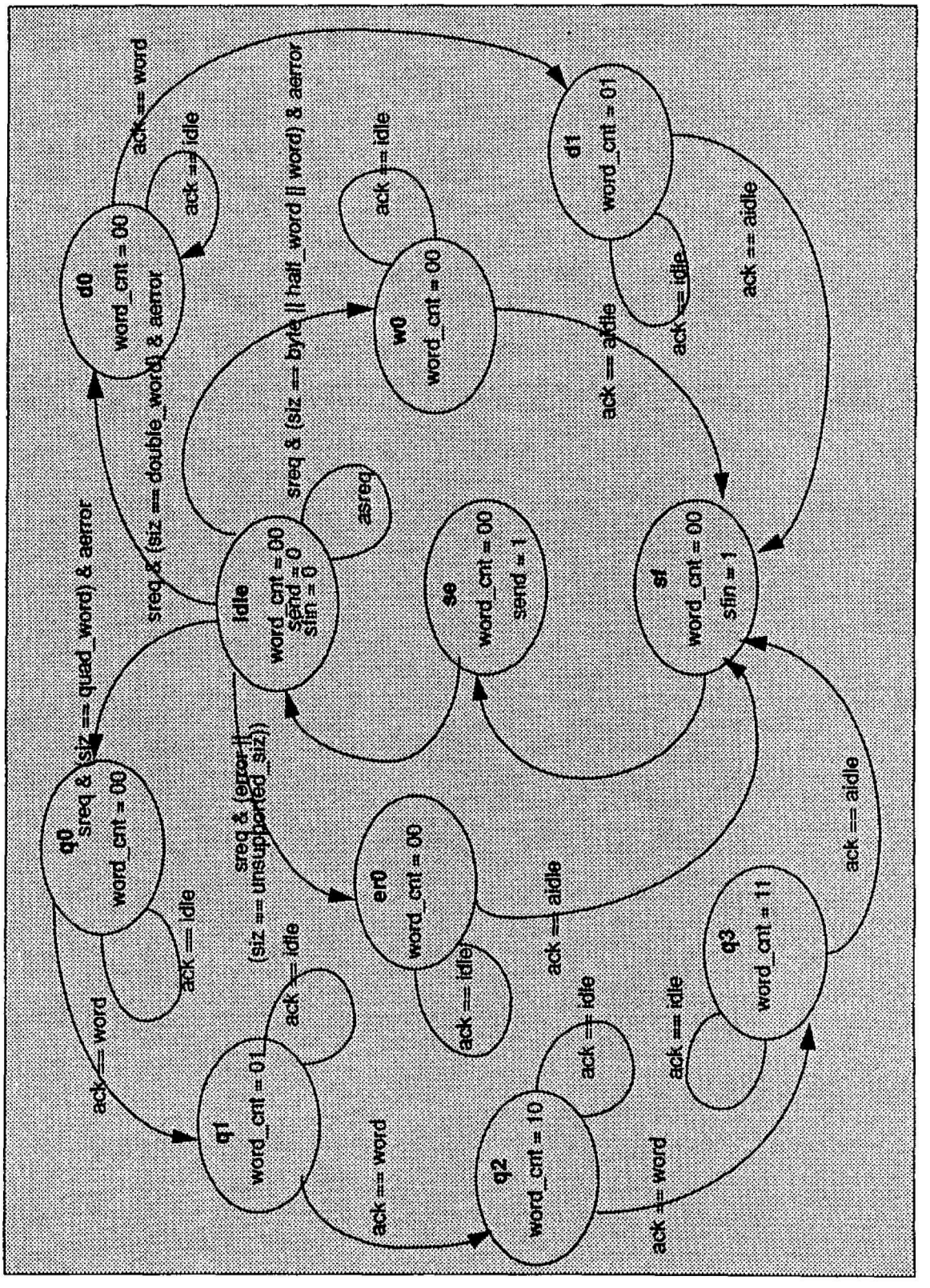
A walk-through of the SSM begins in Idle, where the SBus request line is sampled. Once a request is detected, control transitions to the appropriate state as a function of the bus size and the error signal; W0 if the size is a word or smaller, D0 if the size is a double word, Q0 if the size is a quad word, or Er0 if the size is unsupported or an error was detected. Once in either W0 or Er0, the ACK lines are monitored and any ACK code other than Idle/Wait will cause a transition to Sfin.

Once in D0, the ACK lines are monitored and the SSM is effectively enabled to count Word ACKs. Word ACK will cause a transition to D1. Idle/Wait ACK will keep control in the D0 state. D1 is similar to Er0 and W0. Any ACK code other than Idle/Wait will cause a transition to Sfin.

Once in Q0, (or Q1, or Q2) the ACK lines are monitored Word ACK will cause a transition to D1. Word ACK will bump control to the next higher word count stage until Q3 is reached. Idle/Wait ACK will retain state in Q0 (or Q1, or Q2). Q3 is similar to D1, Er0 and W0. Any ACK code other than Idle/Wait will cause a transition to Sfin.

Once in Sfin, the SBus cycle is nearly complete, except for some amount of housekeeping. Control transitions to Send and then returns to Idle.

Figure 8.7 - S_ctl State Machine

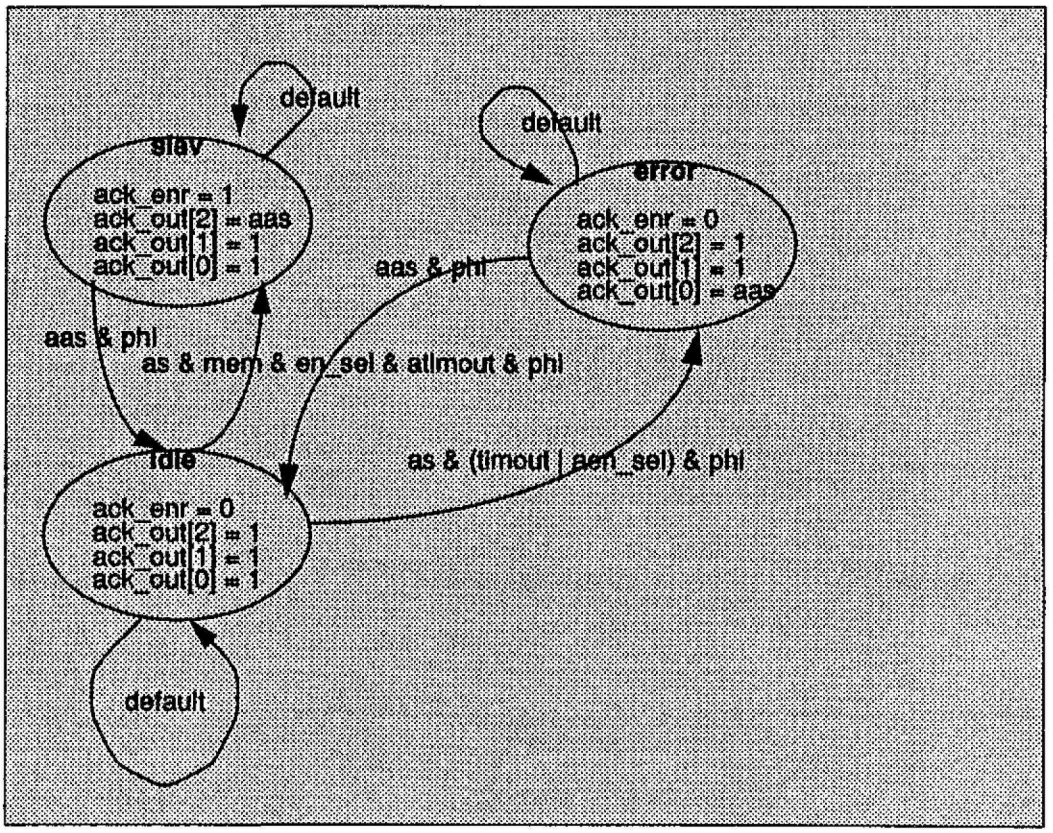


8.0.8 Slave Target Control

The Target control State Machine (TSM) controls the Transfer Acknowledgment (ACK) during local DVMA transactions or error conditions, when it is appropriate for the SBC to drive these signals.

A walk-through of the TSM begins in Idle, where control remains until it recognizes itself as the target of the current slave cycle. Since the TSM is clocked at twice the frequency as the SBus, the phase of SB_CLK is important. If the TSM is the target and either the memory_select or the error signal is detected, then control moves out of Idle to either the Error or Slave state. ACK is enabled and the proper code is asserted. When finished control transitions to the Precharge state where ACK is precharged and then control returns to Idle.

Figure 8.8 - t_ctl State Machine



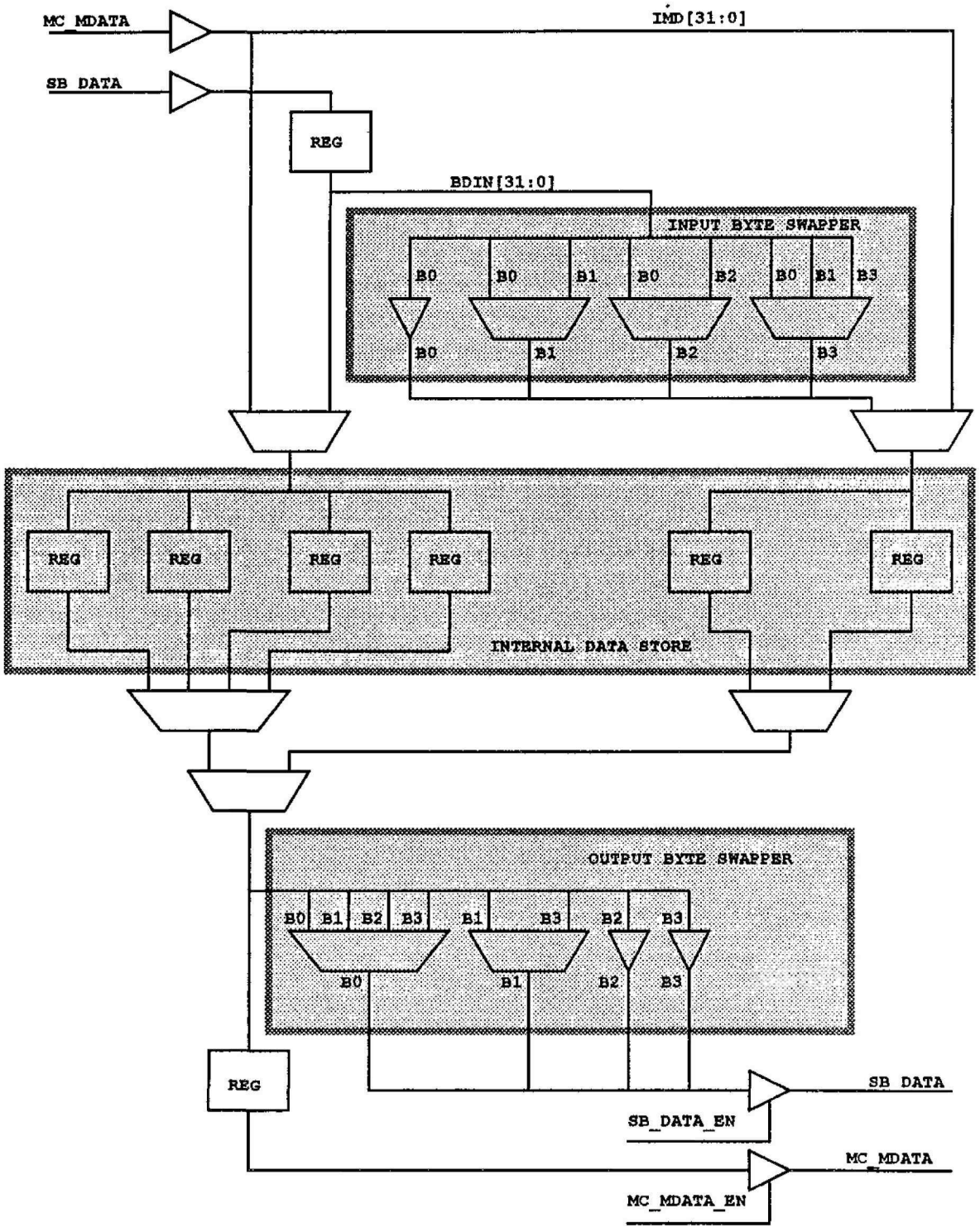
8.0.9 Data Path

The SBC data path consists of a series of multiplexers and registers necessary to transfer data between the SBus devices and local resources. There are two sources of data: the internal data bus, MC_MDATA, which connects local resources to the SBC and the SBus data bus. Data from the SBus is buffered, then passes through a byte swapper, which is necessary to align SBus byte or half-word ports, passes through a source select mux on its way to the internal data store. Data from the internal data bus passes through the source select mux to the internal data store.

The heart of the data path is the internal data store, which provides temporary storage for up to 24 bytes of data. DVMA has exclusive use of 16 bytes of internal storage and 8 bytes are exclusively used for PIO. Each byte-sized register corresponds to an address location. This means that data from a given address location will always be loaded into the same internal data store location, regardless of the order in which the data arrives. Data from either the internal data bus or SBus can to either the DVMA register bank or the PIO register bank.

Data destined for the internal data bus goes from the internal data store, passes through destination select muxes, into an output buffer and is enabled onto the internal data bus by a tristate driver. Data destined for the SBus passes through destination select muxes, through an output byte swapper, necessary to support dynamic bus sizing and is enabled onto the SBus by a tristate driver.

Figure 8.9 - SBC Data Path



8.0.10 Data Control

The SBC data path control logic steers the data through the source and destination multiplexers and loads the data into the internal data store. The source and destination multiplexers are straightforward to control. Since the get and put operations are serial, the data steering is almost static. The main state machine controls the get and put operations.

The internal data store load control works by the application of a mask to the load enables of each byte-sized register. Data can arrive in sizes of as small as a byte. As each piece of data arrives the load enable of its register is masked, thereby preserving the data until the put operation.

8.0.11 Error Handling

The Error Control Block handles errors that occur during both PIO and DVMA transactions. There are three possible sources of error for PIO transactions. There are PIO transactions terminated by timeouts, error acknowledge, and late error. A two-bit error status field, ERR_TYPE, is used to indicate to the CPU the source of error during PIO transactions. This bus is sampled and any code other than that indicating no error signifies that an error has occurred. During this time, the entire state of the current PIO transaction is made available to the CPU for error reporting.

The sources of error for DVMA transactions are translation, parity, timeout and SBus protocol errors. Parity can occur either during address translation or a get operation from local resources. In all cases the SBC becomes the slave target and drives ACK to indicate an error to the DVMA master. Errors during DVMA are transparent to the CPU. The SBC does not use the SBus late error signal to indicate errors

8.0.12 Diagnostic Testing

The SBC employs JTAG and therefore allows all registers to be scanned during JTAG scan mode. Tristate enables for SBus signals that are bidirectional are disabled during scan mode.

A testing feature allows the internal data bus and address bus to be observed when the system is placed in a special diagnostic view mode. When placed in view mode, the SBC steers the internal data bus onto the SBus data bus and the internal address bus onto the SBus address bus. In both cases the address and the data bus information is delayed by one system clock. Of course, the proper SBus address and data is not available during view mode; the SBus is used exclusively for testing at this time.

8.0.13 Additional Work

This section is included for future work. There are two architectural aspects that can potentially yield a better SBus controller design: use a separate, non-unified I/O MMU and a complete handshake protocol between the memory controller and the SBus controller.

PIO and DVMA transactions are distinctly orthogonal operations and as such they can potentially occur in parallel at any time Future designs should look carefully at the costs and benefits of sharing the resources involved (such as unified MMU/IOMMU which precludes some parallelism).

Another optimization that could be made on a future implementation is a full handshake for data transfer between system memory and SBus. This could reduce the amount of internal data storage in the SBC and at the same time increase the transfer sizes that can be supported. If the SBC could request a memory cycle and then signal when data is ready, then the internal data store need only be as big as the data bus size. This is particularly important during DVMA write transactions. With a better handshake mechanism, the SBC could request a memory transfer while the first data word is available. Latency could be avoided and storage elements larger than the size of one data word saved.

9.0 Reset, Clock Control, JTAG

This section will describe the Reset logic, Clock Control logic and the JTAG architecture. The JTAG, reset control and clock start/stop control logic are part of the Misc block, while the clock controller is a design block by itself.

All registers in the microSPARC CPU reset to zero except where otherwise noted. All RAMs including the IU and FPU register files, the data and instruction cache rams, the and TLB remain unchanged by the assertion of Reset.

State and pipeline registers internal to the IU are established on reset via reset logic in the IU, not via explicit reset to the flip-flop. This is to support clearing and setting certain bits (e.g.: S bit of the PSR).

The JTAG logic controls all the scan operation within the chip and in conjunction with the clock start/stop logic, enables the single step operation of the chip for debug purposes. All of the registers in the chip are scannable and are configured as one single internal scan chain for testing as well as debugging the chip.

9.0.1 Reset Controller

The microSPARC Reset Controller performs the simple task of driving microSPARC's internal reset lines, and inhibiting clocks during transitions on those lines to avoid timing violations on the flip-flops being reset.

microSPARC has two reset operations: General Reset (sometimes called SBus Reset) and Watchdog Reset. General Reset is done in response to assertion of the input_reset_l microSPARC input pin; this happens on powerup and on any externally-triggered reset. Watchdog Reset is performed when the IU enters error state due to a taking a trap while the PSR ET bit is deasserted General Reset will cause assertion of both Reset Controller output signals: reset_any and reset_nonwd, Watchdog Reset will cause only reset_any to be asserted. Reset_any resets the IU and any other logic which must be reset only on Watchdog Reset; reset_nonwd resets everything else except the clock and reset logic and the TAP controller.

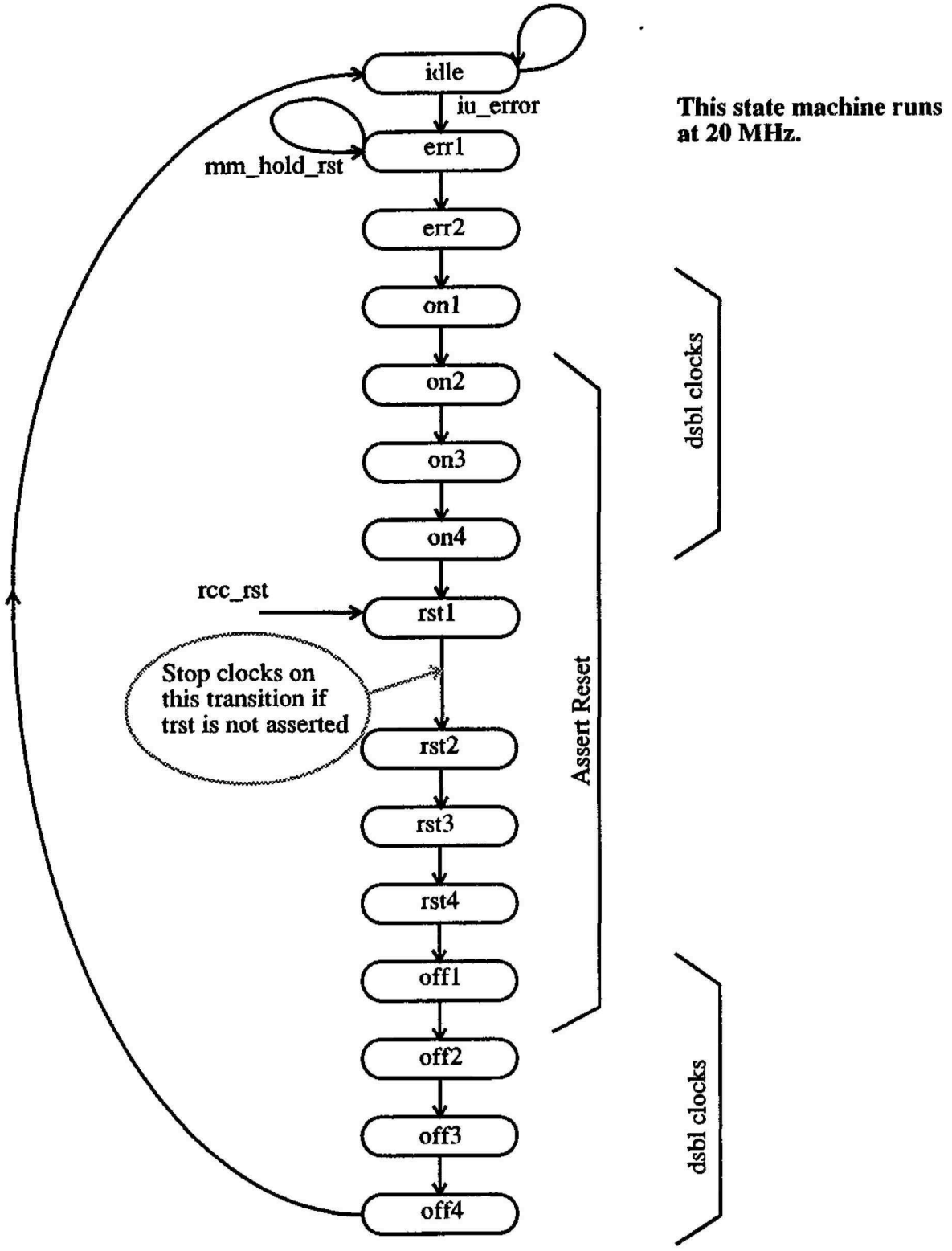
In addition to reset_any and reset_nonwd, the reset controller has another output, rs_dsbl_clocks, which is used to disable the outputs of the clock controller during transitions on the reset lines. This allows the heavily-loaded reset signals time to propagate throughout the chip completely between clocks, to avoid setup and hold time violations. All three of these outputs are controlled by the reset state machine.

However, input_reset_l is combinatorially ORed into both reset_any and reset_nonwd, and rcc_rst forces clocks to be running; taken together, these assure that any circuitry which must (for physical reasons) see reset asserted immediately on powerup will see it (assuming that input_reset_l is asserted, and input_clock is oscillating, immediately on powerup). As a consequence, timing violations may occur on the first clock after assertion of input_reset_l; presumable, the ensuing General Reset will eventually clean up any illegal states caused by these violations.

Inputs which affect operation of the reset state machine are: rcc_rst, a 20-MHz[1] - synchronized version of microSPARC's input_reset_l pin; iu_error, the error state indication from the IU which initiates a Watchdog Reset; and mm_hold_rst, a signal from the MMU which delays the start of a Watchdog Reset sequence until there are no loads, stores, or instruction fetches in progress. Rcc_rst is inhibited during scan shift operations, to prevent loss of non-resettable state if input_reset_l should happen to be asserted during a scan shift.

1. Throughout this section of the document, waveform frequencies and periods will be given as if the frequency of input_clock were 80 MHz, even though this logic will run correctly at any speed from the design frequency (100 MHz) down to DC.

Figure 9.1 - microSPARC Reset State Machine



9.0.2 Reset Controller State Machine Operation

The reset state machine is clocked at 20 MHz. Assertion of rcc_rst synchronously resets the state machine into the rst1 state from any other state. The state machine will thus stay 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/or reset_nonwd are asserted in states on2, on3, on4, rst1, rst2, rst3, rst4, and off1: if the reset sequence was 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 Note that the reset lines transition from 1 to 0 only during a clocks-disabled period, and, for Watchdog Reset, they transition from 0 to 1 only during a clocks-disabled period.

To facilitate scan-based debugging, the reset state machine will assert rs_stop_even upon exiting the rst1 state during a General Reset sequence. If microSPARC's jtag_trst_l input is deasserted at that time, this will cause the clock control state machine to enter the stopped state. The reset sequence will continue as clocks are issued under scan control. It is thus possible to single-step through the remaining states of the reset state machine, and, more importantly, to reset the machine to a known, deterministic state during scan-based debug.

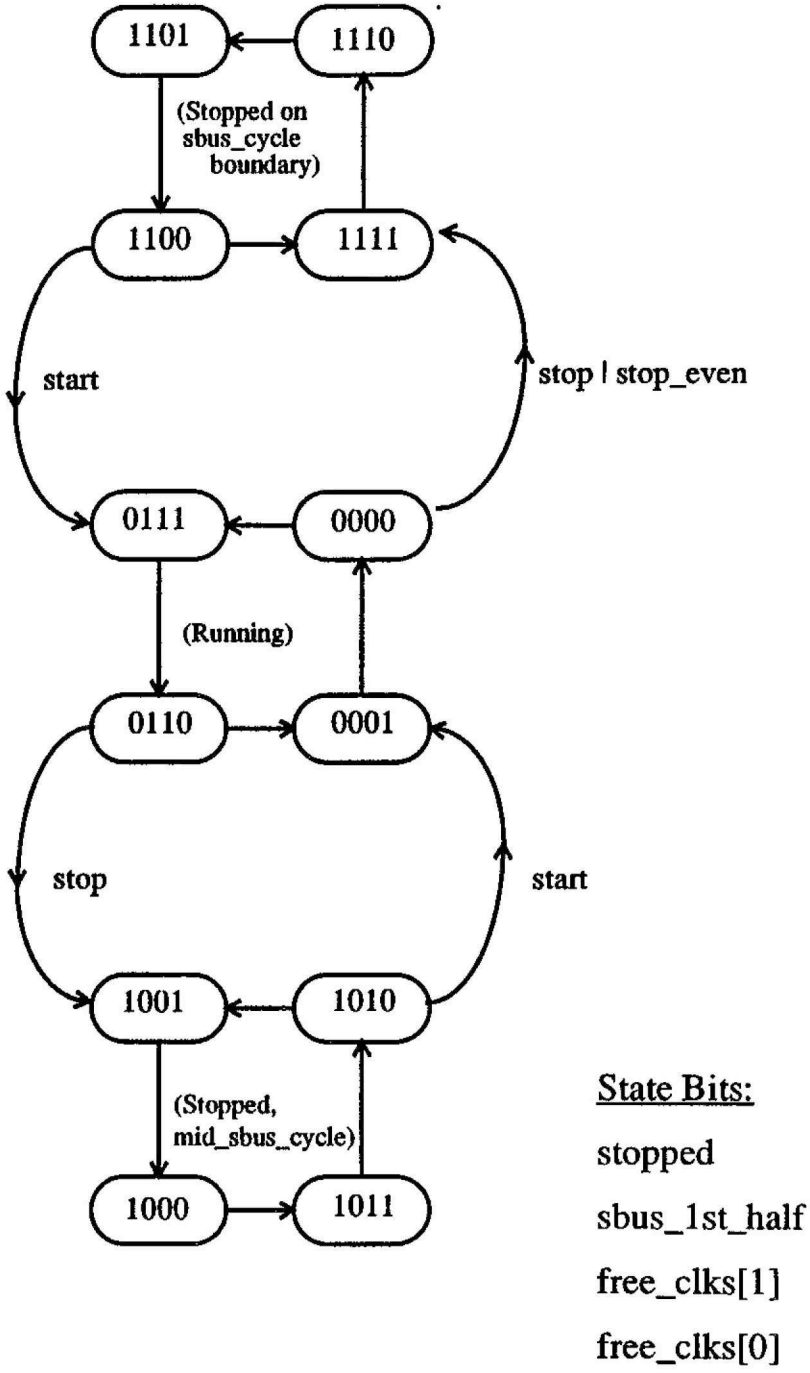
9.0.3 Clock Controller

The microSPARC Clock Controller generates the clock signals used by all of microSPARC (except the TAP controller), as well as the sbclk used by external SBus interface devices. Its operation is controlled by the Clock Control Register (CCR), a collection of internal register bits which are writable only by scan. On Reset, the CCR is cleared. Subsequent scan shift operations can be used to set bits of the CCR in order to alter the operation of clock state machine, as described below. However, the CCR has no effect on the operation of the clock state machine if the jtag_trst_l microSPARC input pin is asserted (low).

At the heart of the clock controller is a 4-bit state machine, clocked by the 80-MHz input_clock. The low-order two bits of this state machine are a free-running two-bit down counter (free_clks[1:0]). The MSB (stopped) indicates whether clocks are stopped or running. The remaining bit (sbus_1st_half) indicates which half of the 20-MHz cycle the state machine is in, even when clocks are stopped The two main clock outputs, ss_clock (40-MHz) and sbclk (20-MHz), are effectively equal to (free_clks[0] | stopped) and (free_clks[1] | stopped), respectively, although in the actual implementation these and all other clock outputs are driven by the Q outputs of 80-MHz-clocked flip-flops. There are three inputs to the clock state machine: start, stop, and stop_even; these are generated in the clk_stop submodule of the misc module. When stop is asserted while the stopped state bit is 0, the clock state machine will take one of these two transitions: 0000->1111 or 0110->1001, whichever comes first. When stop_even is asserted while the stopped state bit is 0, the clock state machine will take the 0000->1111 transition. When start is asserted while the stopped state bit is 1, the clock state machine will take one of these two transitions: 1100->0111 or 1010->0001, whichever comes first. The start input is actually a bit of the CCR, and it will reset itself on the first ss_clock positive edge, to facilitate the single-step operation.

The stopped and sbus_1st_half state bits are readable, but not writable, via scan. Synchronized copies of these two bits form a special two-bit scan chain which may be accessed via the sel_ccr TAP operation. This TAP operation, unlike sel_dbg_scan, does not interfere with the operation of the clock state machine, so the states of these bits may be polled at any time without affecting clocking. Note that 'sel_ccr' is a misnomer, since these two bits are not part of the CCR.

Figure 9.2 - Clock Controller State Machine



9.0.4 Clock Signals

Four distinct clock signals are generated by the clock controller. These are: ss_clock, the 40-MHz signal which clocks most of microSPARC; sbclk, the 20-MHz signal which is driven off-chip to clock the SBus interface logic and the external clock counter, di_val, a half-period-delayed version of ss_clock, used by the cache RAM megacells and other logic which requires a delayed clock; and rcc_clock, a 40-MHz signal which clocks the reset state machine and the CCR logic. All four of these signals will cleanly transition to the high state when the stopped bit of the clock state machine is high. During scan data shift and capture operations, all four clocks are disabled (i.e. forced high) by a synchronized version of the testclken signal sourced by the TAP controller; the clock state machine does not need to be in the stopped state for this to occur. All except sbclk are combinatorially ANDed with testclk (an active-low pulse train generated in the TAP controller by gating jtag_ck) during these disabled periods, so that flip-flops driven by all three of these clocks can be connected together in a single scan chain. All except rcc_clk are disabled by the rs_dsbl_clocks signal sourced by state decodes of the reset state machine, so that slow transitions on the internal reset lines will not cause setup violations. As with the testclken disable, the clock state machine need not be stopped when rs_dsbl_clks is asserted.

9.0.5 Stopping Clocks

To stop clocks, set the stop_clocks CCR bit. This will assert the stop input to the clock state machine, stopping clocks on the next 40-MHz rising edge.

9.0.6 Starting Clocks

To start clocks from a stopped=1 state, set the start bit of the CCR.

9.0.7 Single-Step

From a stopped=1 state, set both the stop_clocks and start bits of the CCR. A single 12.5-ns active-low sys_clk pulse will be issued; if sbus_1st_half was 0, a single 25-ns active-low sbclk pulse will also be issued (its rising edge will coincide with the rising edge of sys_clk

Figure 9.3 - Single Step with sbus_1st_half = 1.

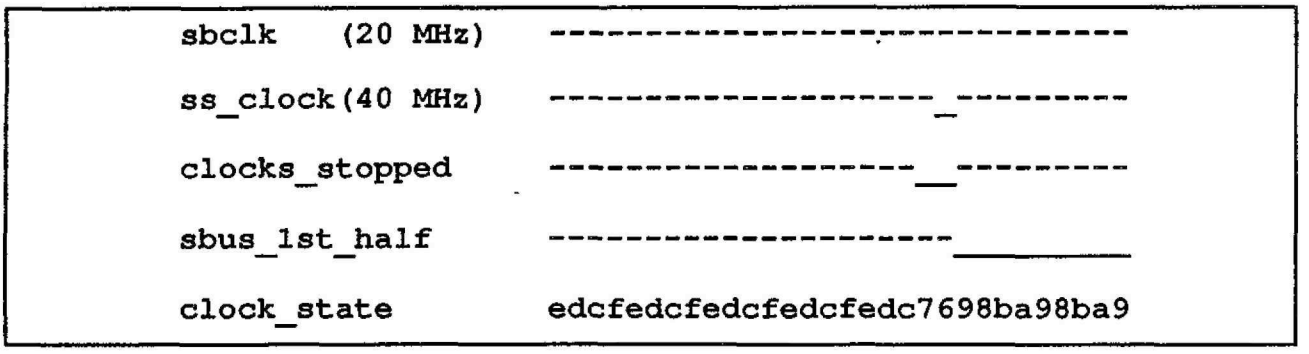
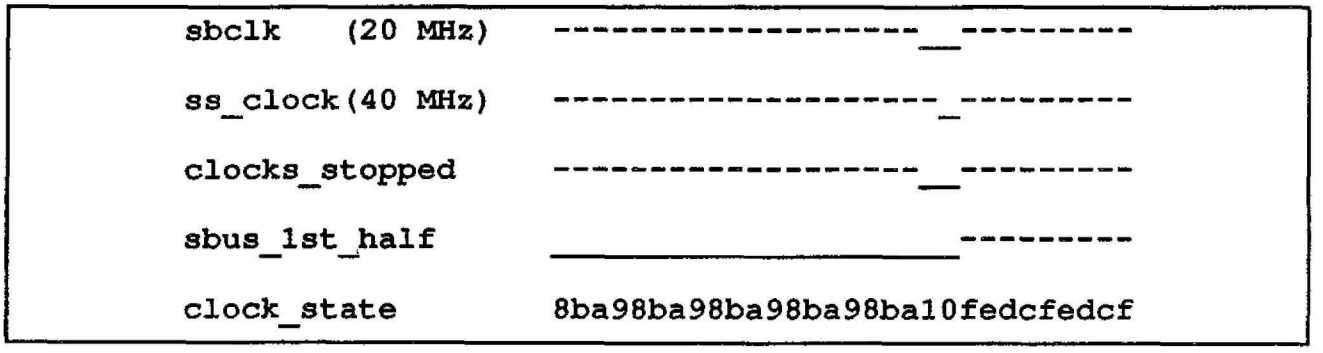


Figure 9.4 - Single Step with sbus_1st_half = 0.



9.0.8 Stop Clocks on Internal Event

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 will stop at the end of the sys_clk cycle in which the input to the int_event flip-flop is asserted. Internal events are detected by special logic in the IU and the MMU - see documentation on those units for more details.

9.0.9 External Cycle Counter

The microSPARC clock controller is designed to interface to a simple external cycle counter (XCC) for precise, at-speed control of system clocking. The interface consists of three microSPARC I/O pins:

* sbclk (output) - the 20-MHz SBus clock output, which is gated off when system clocks are turned off. This output is used to clock the external SBus logic as well as the XCC.

* ext_event (input) - this input is immediately registered in a 20-MHz-clocked flip-flop. Under control of some Clock Control Register (CCR) bits (which are writable only by scan), a logic 1

in this flip-flop will cause clocks to stop either at the next ss_clock rising edge or the next sbclk rising edge. This input should be driven by the terminal_count output of the XCC, perhaps ORed with other externally-detected clock stop signals. In a standard up-counter, the terminal count output is asserted when the counter contains all 1's (i.e. -1).

* int_event (output) - this is the output of a 20-MHz-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 XCC as described below to implement the 'stop N cycles after internal event' function.

Note that this interface runs at the 20-MHz SBus clock rate, and the signal I/O connect directly to inputs or outputs of flip-flops within microSPARC; thus, the XCC logic has nearly a full 50-ns cycle in which to set up its output to the ext_event input.

9.0.10 Counting Clocks

When the XCC is enabled, it increments on every sbclk positive edge. Since the states of the XCC and the CCR are accessible via scan, we can calculate how many 40-MHz system clocks have been issued between any two points in time by scanning out this state information before clocks are started and again after they have been stopped. The following formula can be used. XCC.before and XCC.after are the respective values of the clock counter before and after clocks have been issued, sb1h.before and sb1h.after are the corresponding values of the sbus_1st_half bit of the CCR.

$$N = 2*(XCC.after-XCC.before) - \sim sb1h\ before + \sim sb1h.after$$

This formula of course assumes that XCC has not wrapped around; the XCC control logic should contain a wraparound detector that can be read by scan.

9.0.11 Issuing N Clocks

The XCC can be used to issue exactly N 40-MHz system clocks, at full speed. N can be any number from 1 to approximately 2**(X+1), where X is the number of bits in XCC; for example, a 32-bit XCC lets us control clocks over a 200-second range at 40-MHz operation. This function does not require the use of the int_event output.

Several CCR bits are used for this function. When, while clocks are stopped, a 1 is scanned into stop_on_ext_event and 1 is scanned into start_clocks, clocks will start up and then stop on the next ss_clock rising edge after the ext_event FF goes active; stop_even_on_ext_event is similar to stop_on_ext_event, but it causes clocks to stop on the next *sbclk* rising edge after the ext_event FF goes active. Thus, clocks will stop either one or two 40-MHz cycles, respectively, after a logic 1 is clocked in on the ext_event input. Scan software can scan out the clocks_stopped and sbus_1st_half CCR bits to determine whether clocks are stopped, and if they are stopped in the first or second half of the 20-MHz sbclk cycle.

Figure 9.5 - With stop_on_ext_event

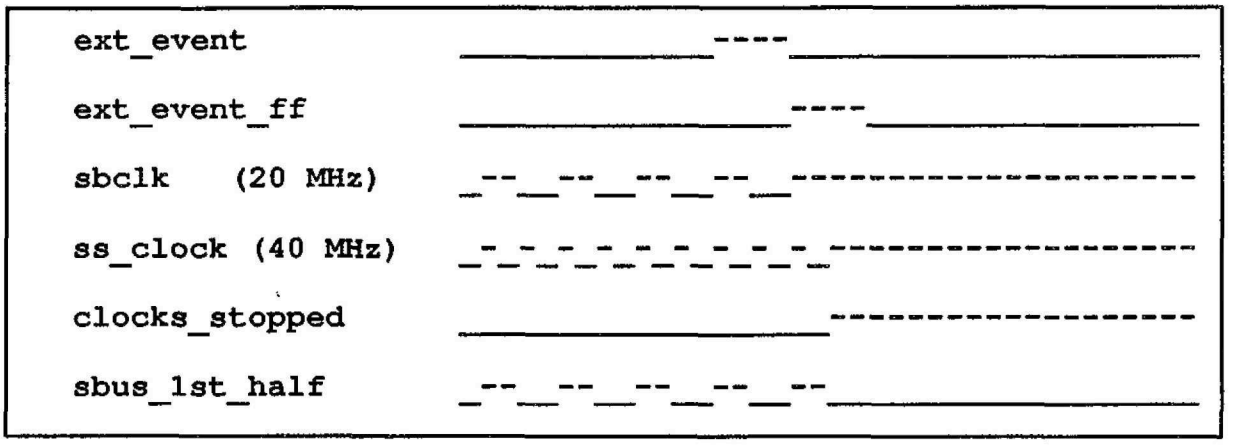
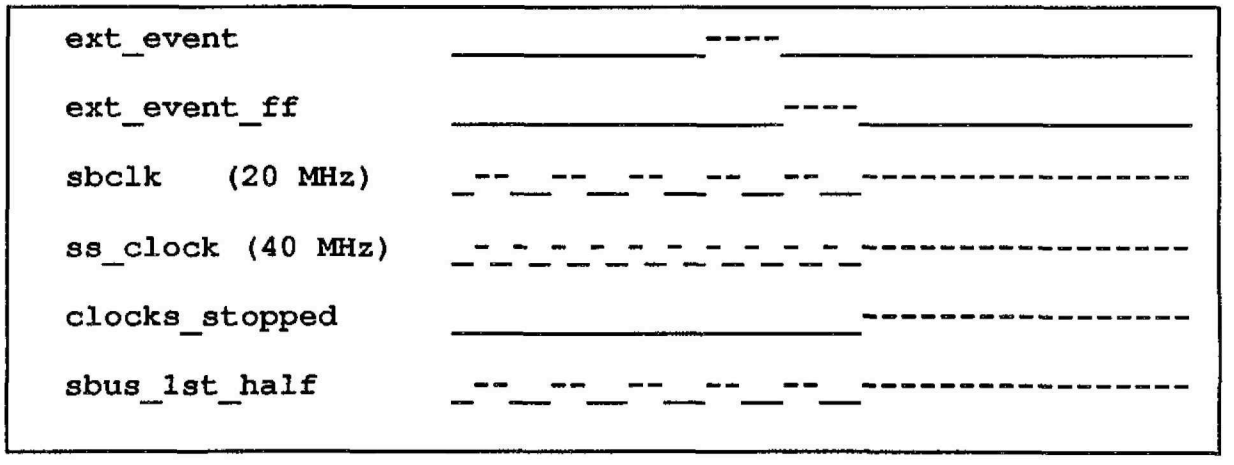


Figure 9.6 - With stop_even_on_ext_event



Scan software can, by scanning appropriate values into the CCR, XCC, and ext_event_ff while clocks are stopped, cause any number of clock pulses to be issued when clocks are restarted, from 1 on up to the maximum. In the table below, 'tc' is the terminal count value of XCC, and M is any integer greater than 1

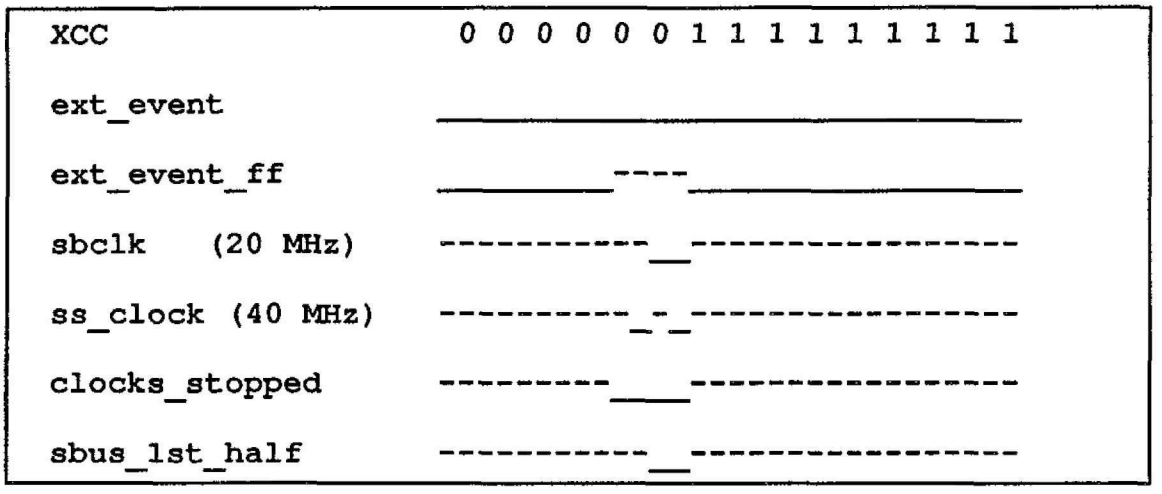
Table 9.1 - Clock Control and Scan

N	sbus_1st_half (scanout)	stop	stop_even (scanin)	ext_event_ff	XCC
1	1	1	0	1	tc+1
1	0	0	1	1	tc+1
2	1	0	1	1	tc+1
2	0	1	0	0	tc
3	1	1	0	0	tc
3	0	0	1	0	tc
2*M	1	0	1	0	tc+2-M
2*M	0	1	0	0	tc+1-M
2*M+1	1	1	0	0	tc+1-M
2*M+1	0	0	1	0	tc+1-M

Examples:

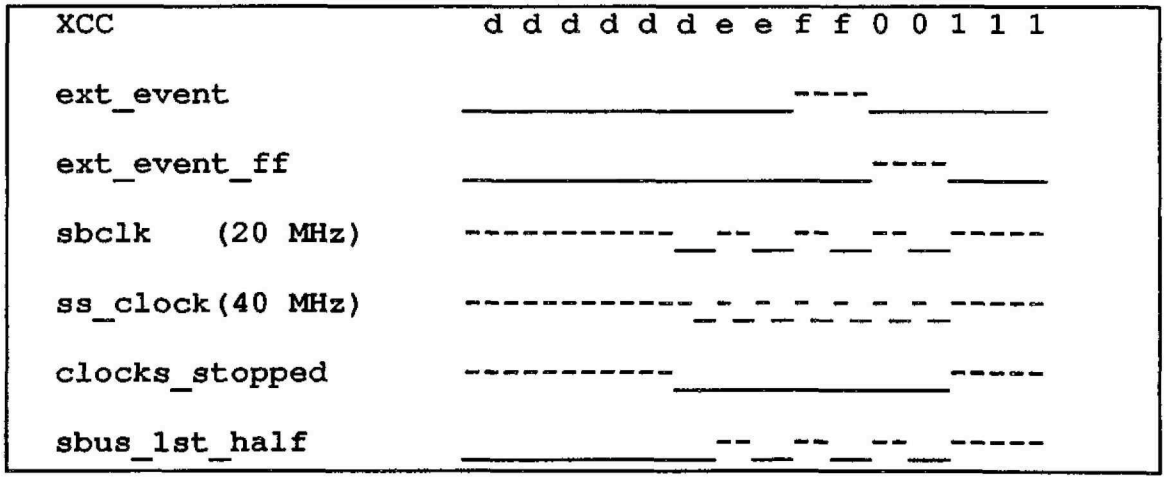
N=2, stopped with sbus_1st_half=1: the table tells us to load (tc+1) into the counter, load 1 into ext_event_ff, and to assert stop_even_on_ext_event. Note that (tc+1)=0.

Figure 9.7 - N=2, stopped with sbus_1st_half=1.



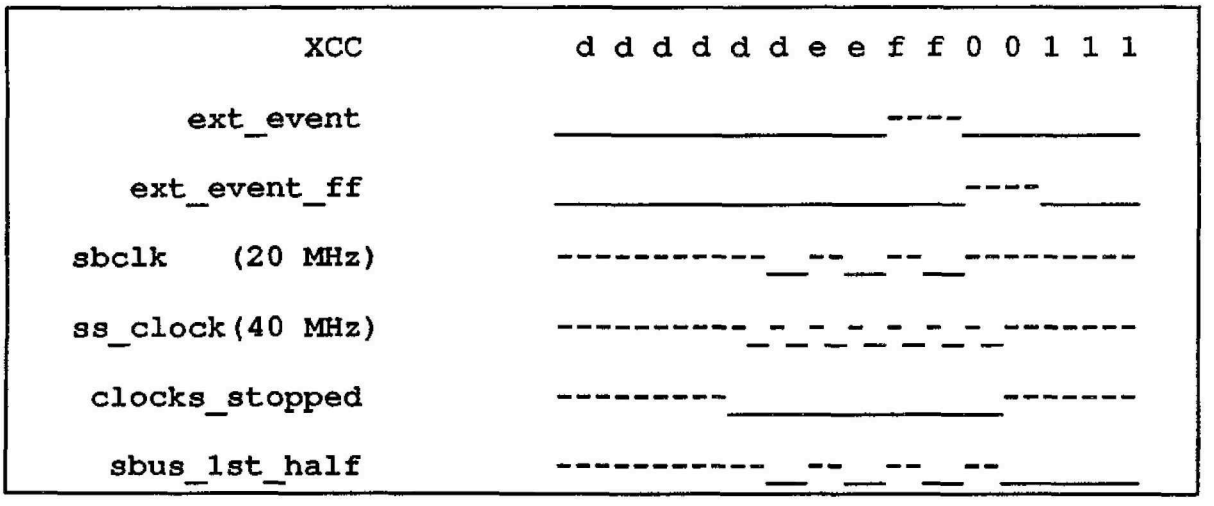
N=7, stopped with sbus_1st_half=0: 7=2*3+1, so M=3. The table tells us to load (tc+1-3)=-3 into the counter, load 0 into ext_event_ff, and to assert stop_even_on_ext_event. I'll show -3 as 'd', which is the last hex digit of its 2's-complement representation.

Figure 9.8 - N=7, stopped with sbus_1st_half=0.



N=7, stopped with sbus_1st_half=1: 7=2*3+1, so M=3. The table tells us to load (tc+1-3)=-3 into the counter, load 0 into ext_event_ff, and to assert stop_on_ext_event. I'll show -3 as 'd', which is the last hex digit of its 2's-complement representation.

Figure 9.9 - N=7, stopped with sbus_1st_half=1.



Note that the last two examples, taken together in sequence, cause 14 positive edges on sys_clk and 7 positive edges on sbclk.

9.0.12 Count Clocks After Internal Event

In this mode, the XCC is held until an internal event occurs. The internal event does not stop clocks, but does cause assertion of the int_event output; the int_event output will remain asserted until it is cleared by scan. The XCC is enabled to count whenever int_event is asserted, so clocks will continue to run until ext_event is asserted, either by XCC or by another external event detector. The intent of this mode is to issue exactly N clocks after the internal event has occurred. Logic in the clock controller records whether the internal event occurred in the first or second half of the bus cycle, and this information is factored into the subsequent clock stop on external event, so that N can be any even or odd integer. Due to latencies in the logic, N must be greater than or equal to 4.

To support this mode, the XCC must have logic which, under scan control, holds the count when int_event is not asserted.

The CCR also needs some logic. The signal int_event_1st_half records whether the internal event which caused the assertion of the int_event

output happened in the first or second half of the SBus cycle. The CCR bit stop_int_to_ext will cause an even or odd number of sys_clk positive edges to occur after the internal event is detected, depending on whether int_to_ext_odd is 0 or 1, respectively. The actual number of clocks issued is (2*(tc-XCC.before) + 4) with int_to_ext_odd=0, and (2*(tc-XCC.before) + 5) with int_to_ext_odd=1. Logic in the clock controller works as follows when stop_int_to_ext is set:

* if int_to_ext_odd=0 and int_event_1st_half=0, then stop clocks at the end of the SBus cycle in which ext_event_ff is asserted (as described above for stop_even_on_ext_event);

* if int_to_ext_odd=0 and int_event_1st_half=1, then stop clocks midway through the SBus cycle in which ext_event_ff is asserted (as described above for stop_on_ext_event);

* if int_to_ext_odd=1 and int_event_1st_half=1, then stop clocks at the end of the SBus cycle in which ext_event_ff is asserted;

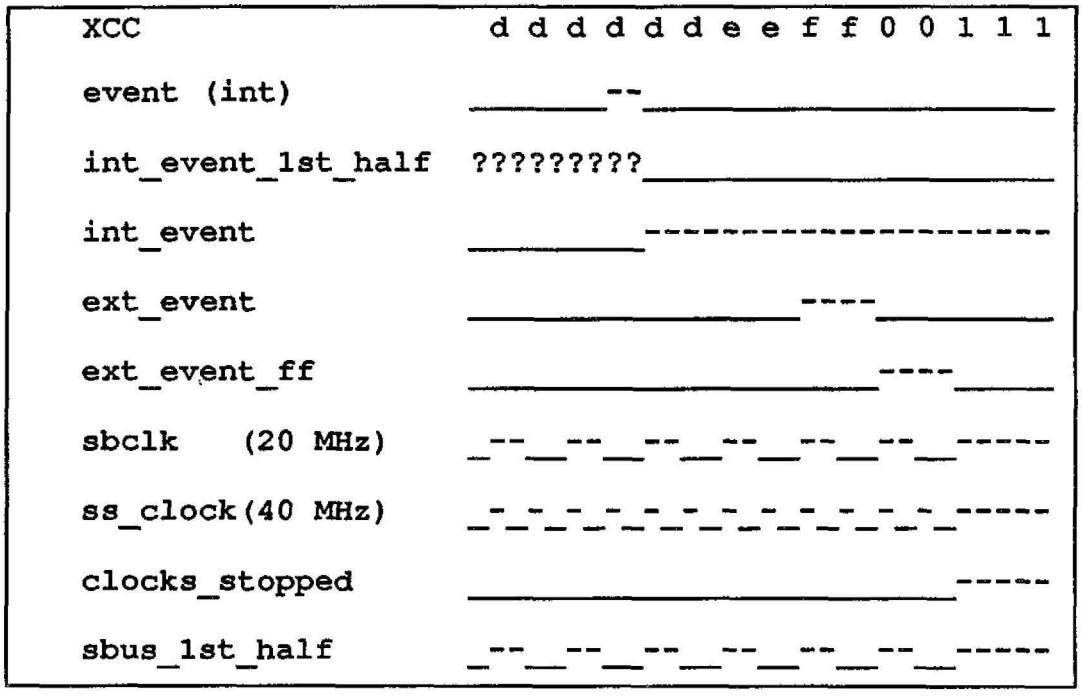
* if int_to_ext_odd=1 and int_event_1st_half=0, then stop clocks midway through the *next* SBus cycle *after* ext_event_ff is asserted.

If, in addtion to setting stop_int_to_ext, we also set stop_on_int_event, then a special mode is enabled. In this mode, as with the simple stop_int_to_ext mode described above, the XCC starts counting clocks after the first internal event, and stops clocks when the count is exhausted. In addition, clocks will stop on an internal event as described in section 2.9.2.5, but only if the internal event occurs while the int_event microSPARC ouput pin is asserted. In other words, while in this mode, clocks will stop on the first internal event which occurs while the XCC is counting; if no such internal event occurs, clocks will stop when the count is exhausted.

Here are some examples:

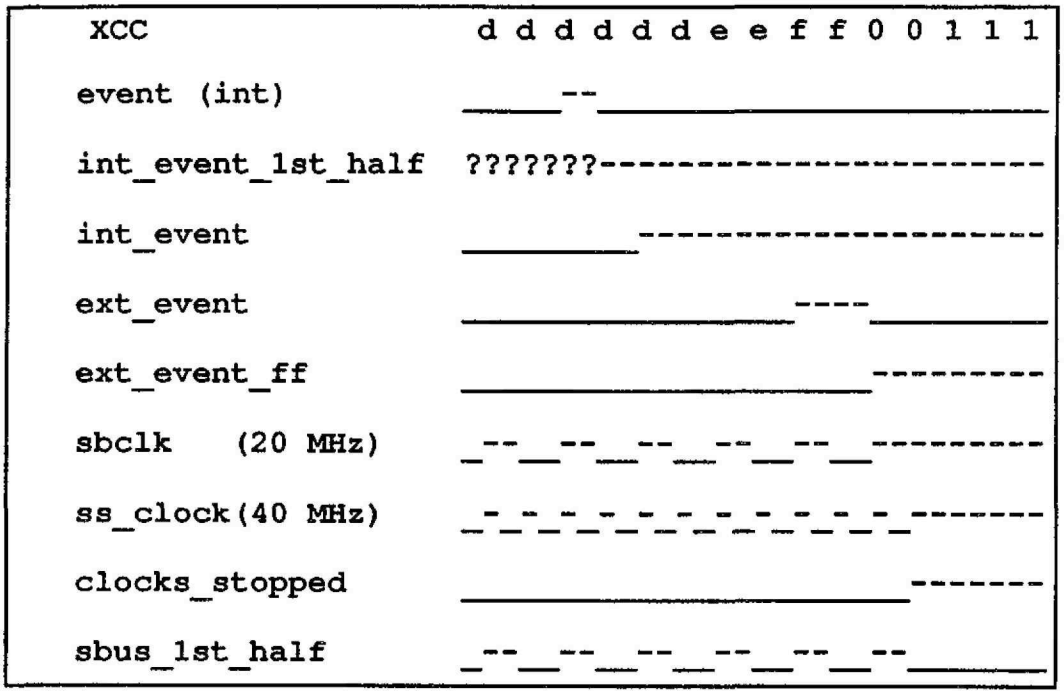
N=8, event occurs in second half of bus cycle. stop_int_to_ext must be set, int_to_ext_odd must be cleared, and XCC.before must be set to (tc-2), here represented by 'd'. Clocks are stopped at the end of the sbclk cycle in which ext_event_ff is active. We get eight more sys_clk rising edges than we would have gotten if clocks had been stopped immediately on the internal event.

Figure 9.10 - Event in First half of bus cycle, N=8.



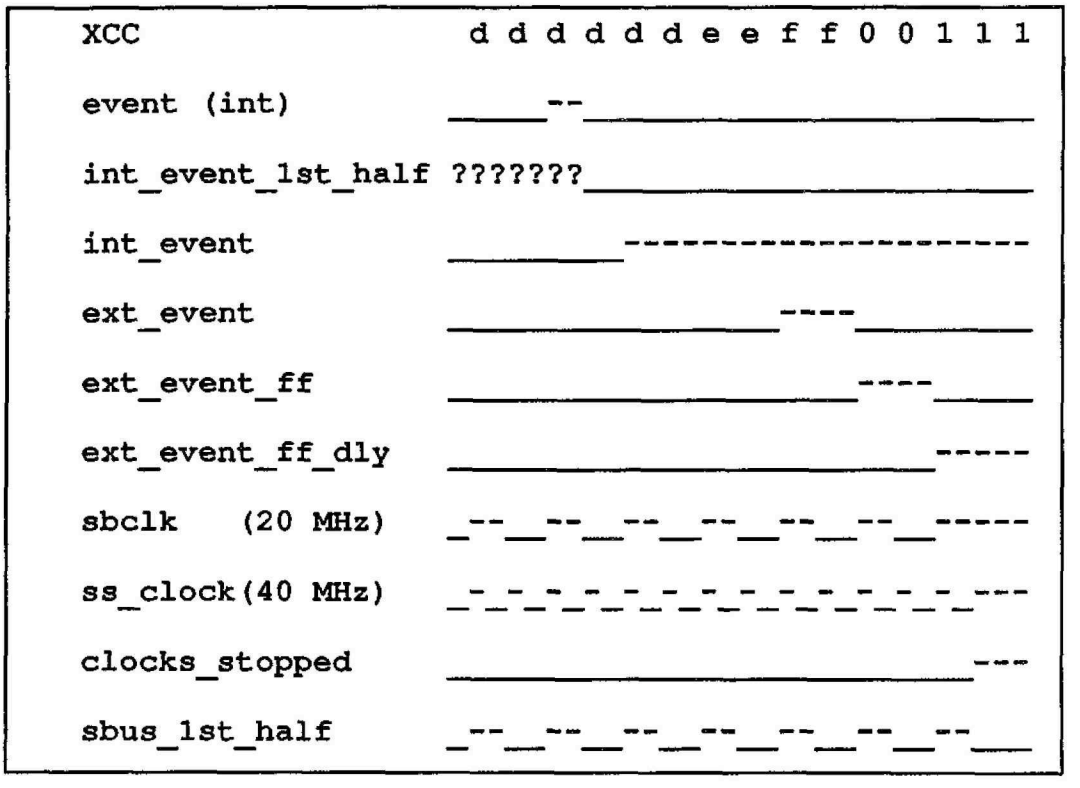
N=8, event occurs in first half of bus cycle. stop_int_to_ext must be set, int_to_ext_odd must be cleared, and XCC.before must be set to (tc-2), here represented by 'd'. Clocks are stopped in the middle of the sbclk cycle in which ext_event_ff is active. We get eight more ss_clock rising edges than we would have gotten if clocks had been stopped immediately on the internal event.

Figure 9.11 - Event in First half of bus cycle, N=8.



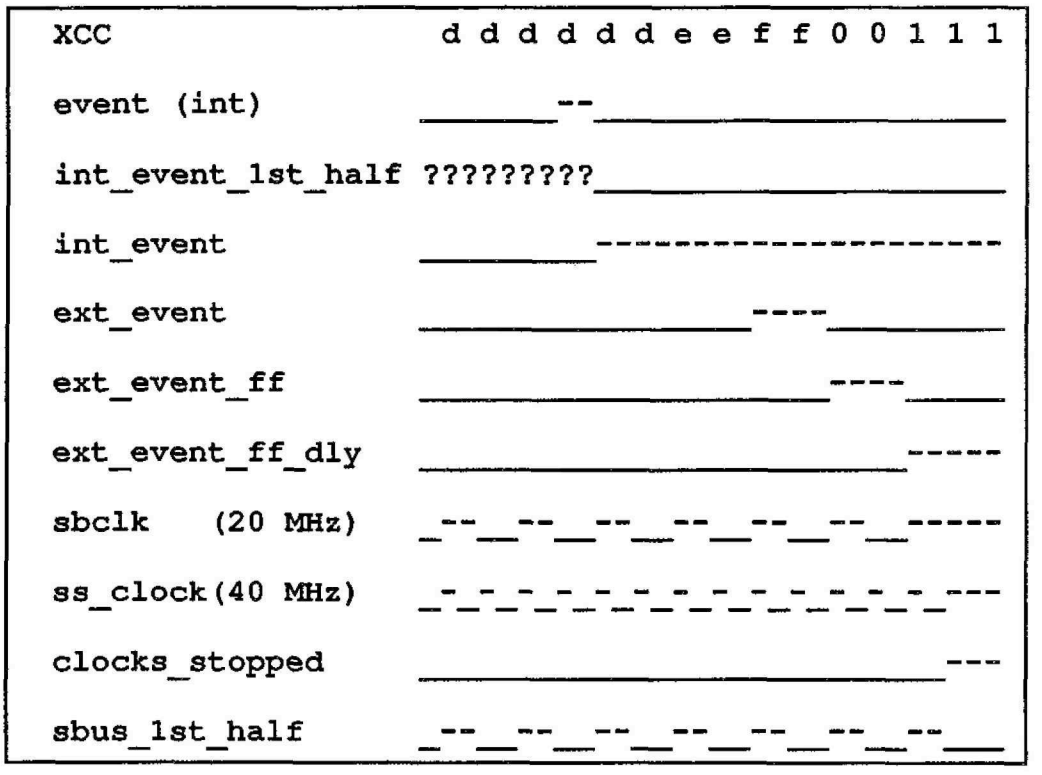
N=9, event occurs in first half of bus cycle. stop_int_to_ext must be set, int_to_ext_odd must be set, and XCC.before must be set to (tc-2), here represented by 'd'. Clocks are stopped at the end of the sbclk cycle in which ext_event_ff is active. We get nine more sys_clk rising edges than we would have gotten if clocks had been stopped immediately on the internal event.

Figure 9.12 - Event in First half of bus cycle, N=9.



N=9, event occurs in second half of bus cycle. stop_int_to_ext must be set, int_to_ext_odd must be set, and XCC.before must be set to (tc-2), here represented by 'd'. Clocks are stopped in the middle of the *next* sbclk cycle *after* ext_event_ff is active. We get nine more sys_clk rising edges than we would have gotten if clocks had been stopped immediately on the internal event.

Figure 9.13 - Event in Second half of bus cycle, N=9.



9.0.13 Stop Clocks After N Internal Events

In this mode clocks are stopped after the Nth sbclk cycle in which the int_event output is asserted. It is controlled by the stop_on_ext_event CCR bit, and XCC needs a scannable control bit which enables it to count only while int_event is active. To use this mode, we must load XCC with (tc-N) and turn on stop_on_ext_event. Latency will be six 40-MHz cycles if the final internal event occurs in the first half of the sbclk cycle, and five cycles if it occurs in the second half. Note that we are not able to handle more than one event per sbclk cycle.

Figure 9.14 - Event in first half of bus cycle, N=2. Latency=6 cycles.

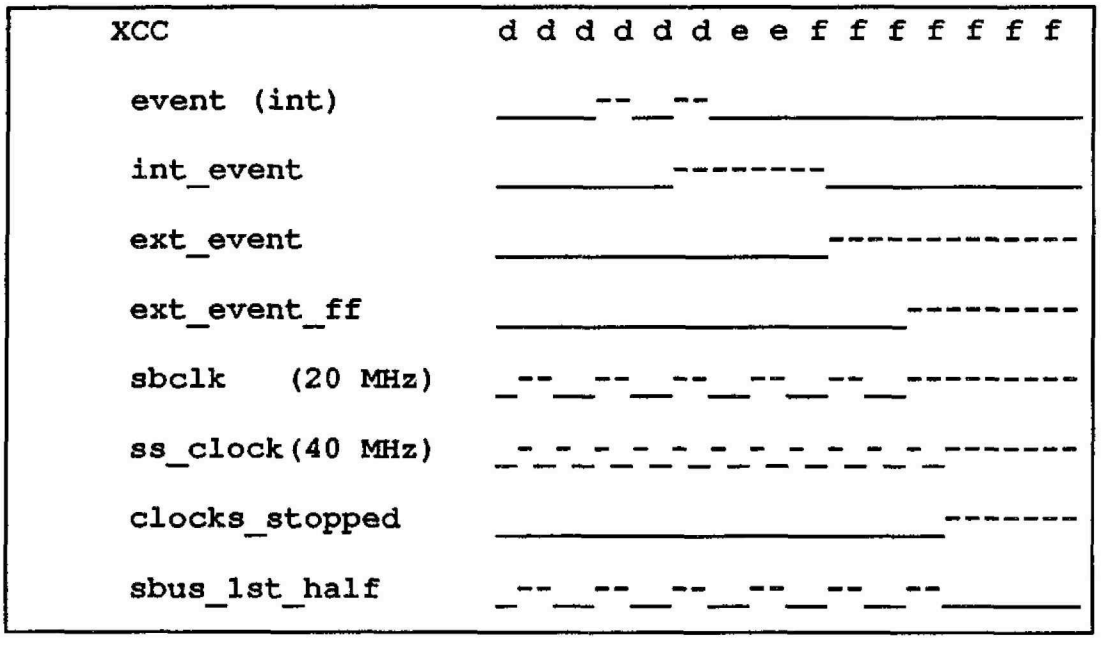
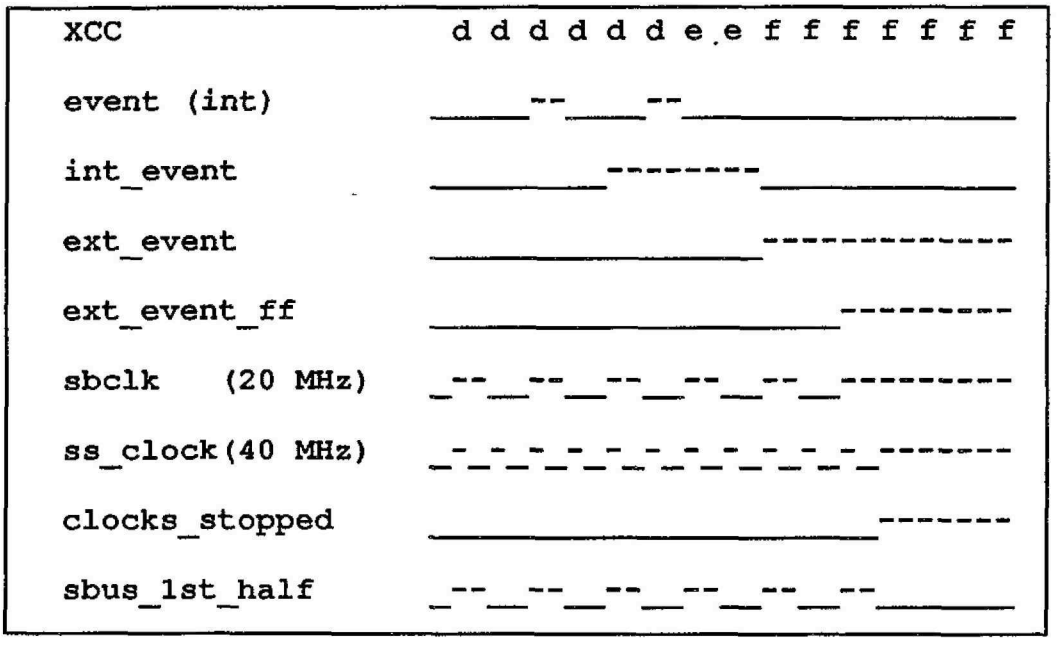


Figure 9.15 - Event in second half of bus cycle, N=2. Latency=5 cycles.



9.0.14 CCR Bits

Here is a list of the Clock Control Register bits. These are accessible by scan only, and their functionality is described above.

*stop_on_ext_event (Issue N Clocks, Stop Clocks after N Internal Events)

*stop_even_on_ext_event (Issue N Clocks)

*stop_int_to_ext (Count Clocks after Internal Event)

*int_to_ext_odd (Count Clocks after Internal Event)

*stop_on_int_event (Stop Clocks on Internal Event)

*stop_clocks (Stopping Clocks, Single-Step)

*start (Starting Clocks, Single-Step)

9.0.15 JTAG

A variety of microSPARC 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 chip (a JTAG slave), via a 4 wire serial testability bus (JTAG bus). The TAP interfaces to the JTAG bus via 5 dedicated pins on the microSPARC 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, please refer to the IEEE document "IEEE Standard Test Access Port and Boundary-Scan Architecture", published by IEEE.

9.0.16 Board Level Architecture

Typical microSPARC systems will contain 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,) which are fed from the JTAG master to all JTAG slaves in parallel, and 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 allows efficient access to any single chip in the daisy-chain without board-level muxing

9.0.17 TAP

The TAP consists of a TAP controller, plus a number of shift registers including an instruction register (IR) and multiple "data" registers.

The TAP controller is a synchronous FSM which 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 that 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 FSM implements the state (16 states) diagram as detailed in the 1149.1 protocol.

The IR is a 6-bit register which allows a test instruction to be shifted into microSPARC The instruction is used to select the test to be performed and/or the test data register to be accessed. The supported instructions are listed in a later section.

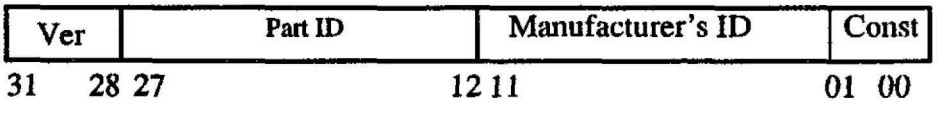
9.0.18 Data Registers

Although any number of loops may be supported by the TAP, the FSM in the TAP controller only distinguishes between the IR and a data register. The specific data register is decoded from the instruction in the IR.

The following data registers are supported in the microSPARC TAP:

* Bypass Register - a single bit shift register for efficient board-level scan.
* Device I.D. Register - a 32-bit register with the following field.

Figure 9.16 - JTAG ID Reg Contents



Field Definitions:

Version - Bits[31:28] represent the version number which is 0x0 for this version

Part ID - Bits[27:12] represent part number as assigned by TI, which is 0x0004

Miff ID - Bits[11:01] represent manufacturer's ID as per JEDEC, which is 0x17

Const - Bit[00] is tied to a constant logic'1'

Value in ID Register. 32'h0000202f

* Data registers - A two bit clock control register to sample outputs from the clock controller(CCR)

* Boundary Scan Register - a single scan chain consisting of all of the boundary scan cells (input, output and inout cells).

* Internal Scan Registers - a single scan chain of all the internal scan f/fs

9.0.19 JTAG Instructions

The following instruction are supported by the microSPARC TAP. The table contains the bit-value and mnemonic, as well as which data register is selected by that instruction. The encodings followed by an "*" are fixed by the IEEE JTAG protocol.

Table 9.2 - JTAG INSTRUCTIONS

value	Name of Instm	Data register(s)	Scan Chains Accessed
000000*	EXTEST	Boundary Scan Register	Boundary Scan Chain
000001*	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 *	BYPASS	Bypass Register	Bypass Register
011110	SEL_CCR	Clock Control Register	Clock Control Register Chain
010000	SEL_INT_SCAN	Internal Scan Register	Internal Scan Chain
011111	SEL_DBG_SCAN	Internal Scan Register	Internal Scan Chain

Note: 1. 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 will be used for ATPG tests, where a clock pulse is needed to capture the next state when 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.

2. The TDO output becomes valid at the falling edge of TCK, per the 1149.1 protocol. This is so, that the TDI input (which is connected TDO

of the preceding component) of the component is stable to be clocked in during the rising edge of TCK.

3. The ATEINTEST operation is used to load the boundary scan f/fs after which, if it enters the 'run_test_idle' state, the JTAG controller will generate a single TCK pulse

Although, we have the capability to single step the chip thru another mechanism (using sys_clock itself), ATEINTEST option provides the capability to perform ICT on the ATE, perhaps at slow speed.

4. The INTEST operation has been added so that it can be used in conjunction with the SEL_INT_SCAN instruction to perform the ATPG test using scan tool. This instruction will not generate any extra clock pulse in run_test_idle state. This is used primarily to load the boundary scan chain.

5. The Sel_CCR is used to sample two bits (stopped, sbus_1st_half) from the clock controller block. These two bits are synchronized (2 stage synchronizer using TCK) before being sampled during the shift-DR state.

9.0.20JTAG Interface to MISC

The JTAG block provides two key signals to the clock controller section, two signals directly to the microSPARC core and a five wire control signal to the boundary scan f/fs.

Clock Controller Interface:

Testclk and Testclken are the two signal that are generated in the JTAG block and sent to the clock controller.

Testclken is an active high signal that switches the ss_clock (the 40MHz) to the core from the normal 40MHz clock to the Testclk.This happens only for certain JTAG instructions They are:

sel_int_scan, sel_dbg_scan, intest, ateintest

For all other instructions (extest, sample, bypass, idcode, sel_ccr) testclken remains inactive thus enabling the normal 40 MHz clock to microSPARC core. The Testclken signal is synchronized inside the clock controller using the free_20MHz clock. By design Testclken is generated to be active at least three TCK cycles before the Testclk signal becomes active. Testclken signal changes state only after transition

from update_IR of the instruction scan cycle, on the positive edge of TCK. Testclken signal becomes inactive after transition to tap_logic_reset state on the falling edge of TCK.

Testclk is a gated version of TCK and the gating signals are sel_instruction and shift (function of shift_DR) and capture (capture-DR) states. Testclk toggles only during sel_int_scan and sel_dbg_scan instructions.

microSPARC Core Interface:

Sys_sen (ss_scan_mode) and tg_strobe are two signals that go directly to the core of microSPARC. Scan_mode signal is active high whenever the Tap enters any of the four DR states, shift,exit1,pause and exit2. During the last three state, Testclk will not toggle and the state of the f/f remains the same as the last bit scanned in during the shift state. It is necessary to activate the scan_mode signal during these three states, so that tri-states would remain disabled during repeat scan after going thru exit1, pause, exit2 states. Sys_sen is a registered signal that is clocked on the falling TCK. This has been done to avoid race conditions between the scan_mode signal and the shift clock(testclk) during the shortest tap state traversal from select-DR to shift-DR.

Since the Sys_sen is a heavily loaded (goes to all f/fs in the chip) signal, it may have a longer rise time and not meet the setup time requirement for the shortest tap state traversal from select-DR to shift-DR. In such a case, the TCK should not be run at greater than 5 MHZ.

The tg_strobe signal is low going pulse that is used as a self-timing trigger for the megacells. It is generated during the update-DR state and adheres to the timing specified in the megacell document.

Boundary Control Interface:

The five wire boundary control signal corresponds to: bin_cap, bout_cap, b_sen, b_uen, b_mode.

bin_cap and bout_cap are generated during the capture-DR state and are used to load the value on the pins or the output of the core to the boundary scan f/f b_sen is generated on the falling edge of the tck (to avoid race conditions) and is used as a scan_en signal for the boundary scan f/f. b_uen is an update signal for the boundary scan update latch and it happens 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 will change

during the update-IR state and when the tap goes back to test-logic-reset state on the falling edge of TCK.

RESET Mechanism.

We also have a independent TRST_L signal which when active low would set the TAP into the tap_logic_reset state. This signal will asynchronously set the tap state machine to the tap_logic_reset state. It adheres to the 1149.1 IEEE protocol with respect to the initialization thru reset mechanism. There is no minimum active time requirement on this reset signal. If the board is not going to have an extra oscillator for TCK, then the JTAG reset pin (TRST_L) can be tied to an active low signal thus disabling JTAG operations in the chip.

The TDI and TMS inputs have pullups on the pad and when left unconnected will be equivalent to a signal value '1' on these pins. With a free running TCK, it would guarantee that the TAP would get into the tap_logic_reset state at the end of five TCKs.

9.0.21JTAG Operation

The following are some of the basic operations which, when combined together will enable the user to run any of the JTAG instructions specified above. They are provided here just for understanding the TAP state transitions during various JTAG operations.

We will only be concerned with JTAG I/O, i.e. TCK, TMS, TDI, TRST and TDO The first four are inputs and the last one is the output. All five are chip I/O. The other inputs to the chip are either in a don't care state or in a predetermined state. They shouldn't affect the operation of the JTAG controller. It should be noted, that, for a more robust operation of the chip, we should follow a proper procedure with regard to getting in and out and back to JTAG operations.(foi instance resetting the system before and after JTAG operations. Once we are in the tap_logic_reset state, all outputs from JTAG become inactive and the chip should be back to normal functional mode.)

The tap state encodings (in hex) are as follows:

f-*test-logic-reset*, c-*run-test-idle*, 7-*select-DR*, 6-*capture-DR*, 2-*shift-DR*, 1-*exitl-DR*, 3-*pause-DR*, 0-*exit2-DR*, 5-*update-DR*, 4-*select-IR*, e-*capture-IR*, a-*shift-IR*, 9-*exit1-IR*, b-*pause-IR*, 8-*exit2-DR*, d-*update-IR*

In order to run the JTAG instructions, we do the following tap state traversal for the various sub tasks:

Instruction Scan:

f -->c -->7 -->4 --> e -->9 -->b -->8 -->a (for 6 clocks) --> 9

(the opcode is shifted thru tdi while in the shift-IR state)

Data Scan:

9 -->b --> 8 --> d --> c -->7 -->6 -->1 --> 3 -->0 -->2(# of

shifts equal to length of scan chain) --> 1

(At state 'd' the decode instruction is latched on the falling edge of tck. Data is shifted into appropriate data register during shift cycle and at the end of shift exit to exit1-DR(1) state.

Return to new instruction:

2 --> 1 -->3 --> 0 --> 5 --> c

(we will wait in state c (run-test-idle) and go back to instruction scan as shown above.)

Figure 9.17 - JTAG LOGIC BLOCK DIAGRAM

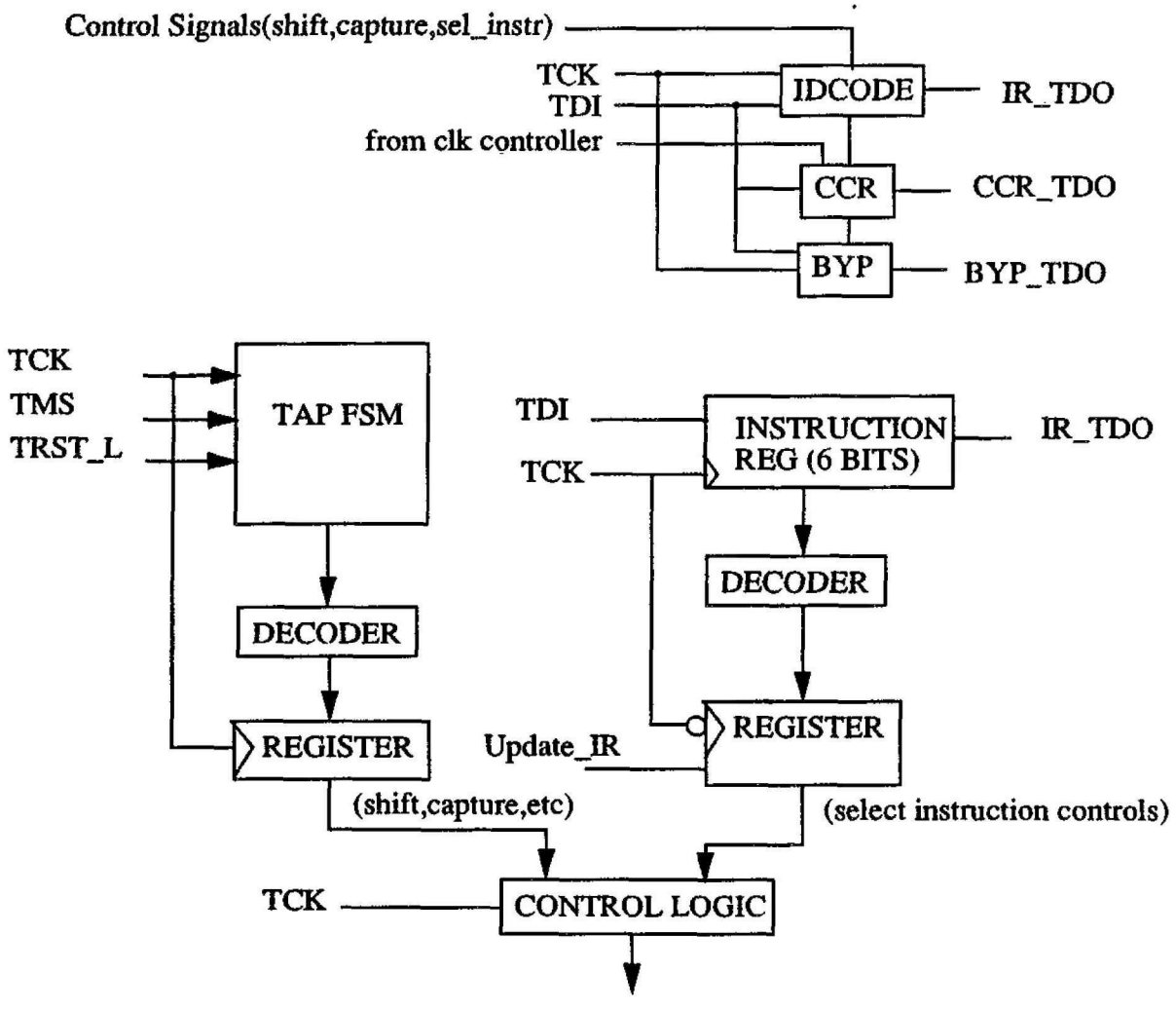
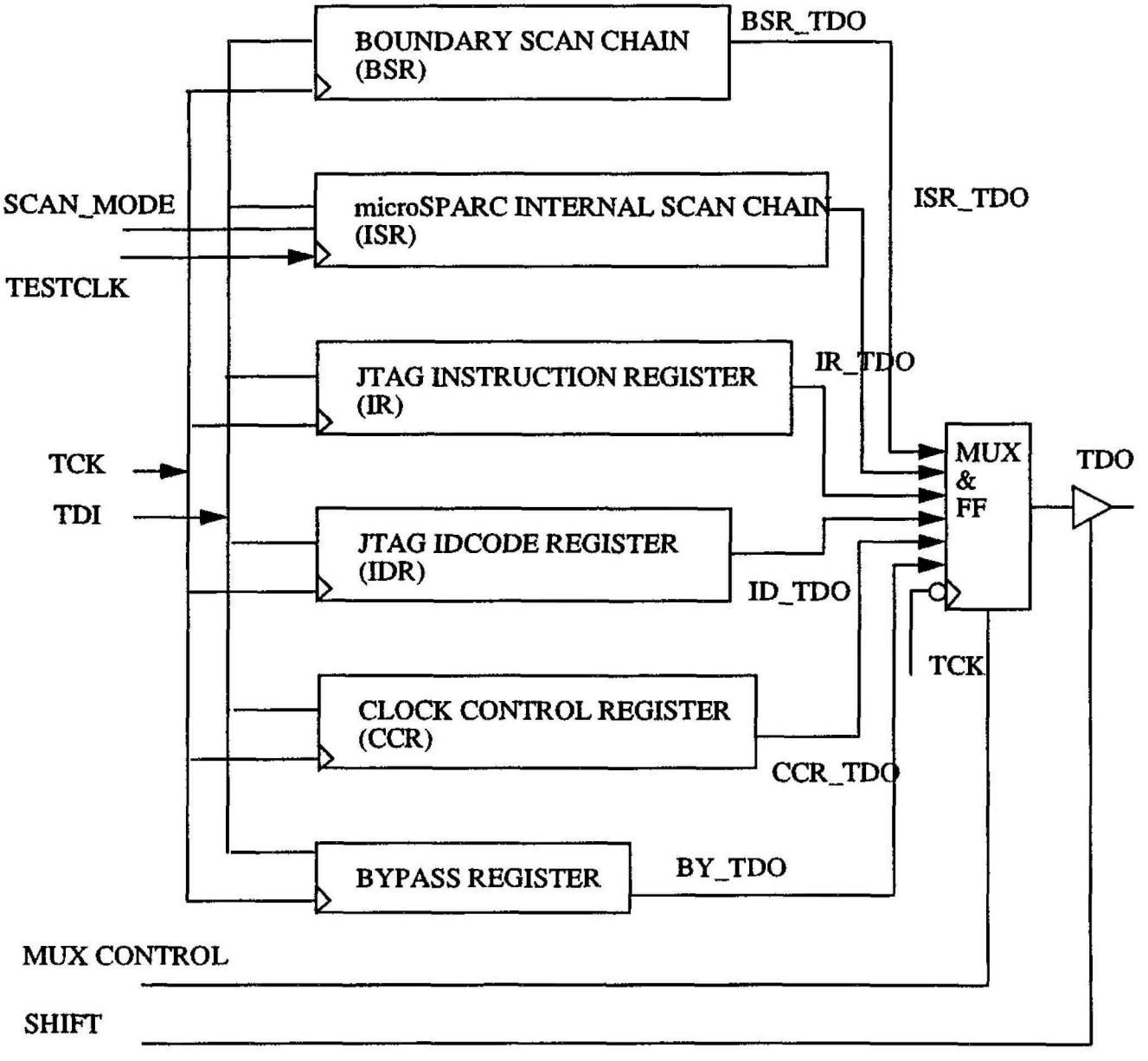


Figure 9.18 - microSPARC JTAG DATA & INSTRUCTION REGISTERS



10.0 Error Handling

The microSPARC CPU must detect and handle many kinds of errors and exceptions. The SPARC IU is interrupted by some type of trap in all CPU error cases. DMA masters other than the CPU should cause their own IU trap via the SBus interrupt mechanism. Physical address references to nonexistent addresses in any address space will either return garbage or cause timeouts. The following preliminary list attempts to describe what happens under various circumstances.

Table 10.1 - 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)
	IO DMA Read Memory Access	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
	IO DMA Write Byte, Half-word Memory Access (Read-modify-write)	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
	Tablewalk on IO DMA Memory Access	return SBus Error Acknowledge set PE, ERR in MFSR save PA in MFAR, cause L15 interrupt
SBus Controller Time Out	CPU SBus Read Access	set TO, FT=5, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
	CPU SBus Write Access	set TO, ERR, SIZE, ~RD, FAV in AFSR save PA in AFAR cause L15 interrupt
	IO DMA Access	return SBus Error Acknowledge
SBus Late Error (Ack)	CPU SBus Read Access	set LE, ERR, SIZE, RD, FAV in AFSR
	CPU SBus Write Access	set LE, ERR, SIZE, FAV(sometimes) in AFSR

Error	Initiator	Result Summary
SBus Error Acknowledge	CPU Read Access	set BE, FT=5, FAV in SFSR save iu_dva in SFAR cause Data Access Error trap (R stage)
	CPU Write Access	set BE, ERR, SIZE, ~RD, FAV in AFSR, save PA in AFAR cause L15 interrupt using CP_STAT
Invalid Address Error	IO DMA PTE Access (IO PTE V bit = 0)	return SBus Error Acknowledge
	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)
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)
Protection Error (Write to read only page)	IO DMA Write	return SBus Error Acknowledge

11.0 ASI Map

This chapter describes the microSPARC ASI map.The Address Space Identifier (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, whether an access is to instruction or data memory, and is used to perform other internal cpu functions

11.0.1 Overview

The table below lists all of the ASI values supported in a microSPARC system. Only the least significant 6 bits of the ASI are decoded.

Table 11.1 - ASI's Supported by microSPARC

ASI	Function	Access	Size
00	Reserved	-	-
01-02	Unassigned	-	-
03	Ref MMU Flush/Probe	Read/Write	Single
04	MMU Registers	Read/Write	Single
05	Unassigned	-	-
06	Ref MMU Diagnostics	Read/Write	Single
07	Unassigned	-	-
08	User Instruction	Read/Write	All
09	Supervisor Instruction	Read/Write	All
0A	User Data	Read/Write	All
0B	Supervisor Data	Read/Write	All
0C	Instruction Cache Tag	Read/Write	Single
0D	Instruction Cache Data	Read/Write	Single
0E	Data Cache Tag	Read/Write	Single
0F	Data Cache Data	Read/Write	Single
10-14	Unassigned	-	-
15-16	Reserved	-	-
17-1C	Unassigned	-	-
1D-1E	Reserved	-	-
1F	Unassigned	-	-
20	Ref MMU Bypass	Read/Write	All
21-2F	Reserved	-	-
30-35	Unassigned	-	-
36	Instruction Cache Flash Clear	Write	Single
37	Data Cache Flash Clear	Write	Single
38	Unassigned	-	-
39	Data Cache Diagnostic Register Access	Read/Write	Single
3A-3F	Unassigned	-	-
40-FF	Reserved	-	-

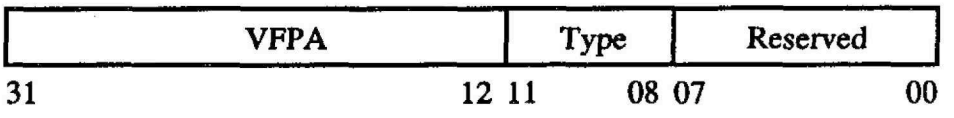
ASI Descriptions:

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 **Ref MMU Flush/Probe** - This space is used for a flush or probe operation. The Virtual Address is decodes as follows.

Figure 11.1 - TLB Flush or Probe Address Format



Field Definitions:

Virtual Flush or Probe Address (VFPA) - This field is the address that is used to index into TLB. Depending on the type of flush or probe not all 20 bits are significant.

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.

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. Note that any TLB flush also flushes the ITBR automatically

Table 11.2 - TLB Entry Flushing

VA[11:08]	Flush	PTE Match Criteria
0	Page	(Level 3) AND (Context match OR ACC=6-7) AND VA[31.12] match
1 to 4	Entire	None (Entire TLB Flush)
5 to F	Reserved	

Probes cause the MMU to perform a table walk stopping when a PTE has been reached as shown.

Table 11.3 - CPU TLB Entry Probing

VA[11:08]	Probe	Returned Data
0	Page	Level 3 PTE or 0
1	Segment	Level 2 PTE or 0
2	Region	Level 1 PTE or 0
3	Context	Level 0 PTE or 0
4	Entire	PTE from Table Walk or 0
5 to F	Reserved	

ASI=0x04

MMU Registers - This space is used to read and write internal MMU registers using the Virtual Address to reference them. Single word accesses only should be used, others result in an error.

Table 11.4 - 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**
14	Synchronous Fault Address Register**
15-1F	Reserved
	**Writeable 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 **Ref MMU Diagnostics** - Diagnostic reads and writes can be made to the 32 TLB entries and the Instruction Translation Buffer Register 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.

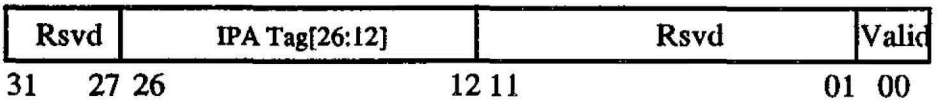
ASI=0x09 **Supervisor Instruction** - This space is defined and reserved by SPARC for supervisor instructions.

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** - This space is used for reading and writing instruction cache tags by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x0FFF on modulo-32 boundaries.

Figure 11.2 - Instruction Cache Tag Entry



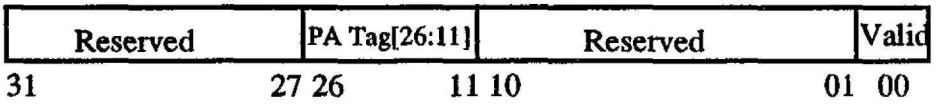
Bits [31:27,11:01] are not implemented, should be written as 0 and will be read as 0.

ASI=0x0D **Instruction Cache Data** - This space is used for reading and writing instruction cache data by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x0FFF.

ASI=0x0E

Data Cache Tag - This space is used for reading and writing data cache tags by using the LDA and STA instructions at virtual addresses in the range of 0x0 to 0x03FF on modulo-16 boundaries.

Figure 11.3 - Data Cache Tag Entry



Bits [31:27,10:01] are not implemented, should be written as 0 and will be read as 0.

ASI=0x0F

Data Cache Data - This space is used for reading and writing data cache data by using the LDA and STA instructions in ASI 0xF at virtual addresses in the range of 0x0 to 0x03FF.

ASI=0x10-0x14

Unassigned - This space is unassigned and may be used in the future.

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

Ref MMU Bypass - This space can be used to access an arbitrary physical address. It is particularly useful before the MMU or main memory have been initialized. The MMU does not perform an address translation rather 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=0x30-0x35 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x36 **Instruction Cache Flash Clear** - The instruction cache is completely flushed by any type of alternate store instruction to this ASI. All instruction cache valid bits are reset (to zero) by this operation. Note that the pipeline is NOT flushed by this sta as it would be on a SPARC FLUSH instruction.

ASI=0x37 **Data Cache Flash Clear** - The data cache is completely flushed by any type of alternate store instruction to this ASI. All data cache valid bits are reset (to zero) by this operation.

ASI=0x38 **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x39 **Data Cache Diagnostic Register Access** - This space is used to read and write the internal Data Cache Registers. iu_dva[08] is also used to select from between WRB0 and WRB1. Single word accesses only should be used, others result in an internal error. The Virtual Address map to these registers:

Table 11.5 - Address Map for Data Cache Registers

VA[08]	Register
0	Write Buffer 0
1	Write Buffer 1

VA bits [31:09] are zero. VA bits [07:00] are ignored and should be set to zero by software.

ASI=0x3A-0x3F **Unassigned** - This space is unassigned and may be used in the future.

ASI=0x40-0xFF **Reserved** - Since the 2 high order bits are not decoded these encodings should not be used.