

U2S User's Manual

Sun Microelectronics

U2S User's Manual

©1997 Sun Microsystems, Inc. All rights reserved.

THE INFORMATION CONTAINED IN THIS DOCUMENT IS PROVIDED "AS IS" WITHOUT ANY EXPRESS REPRESENTATIONS OR WARRANTIES. IN ADDITION, SUN MICROSYSTEMS, INC. DISCLAIMS ALL IMPLIED REPRESENTATIONS AND WARRANTIES, INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

This document contains proprietary information of Sun Microsystems, Inc. or under license from third parties. No part of this document may be reproduced in any form or by any means or transferred to any third party without the prior written consent of Sun Microsystems, Inc.

Sun, Sun Microsystems and the Sun Logo are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. in the United States and other countries. Products bearing SPARC trademarks are based upon an architecture developed by Sun Microsystems, Inc.

The information contained in this document is not designed or intended for use in on-line control of aircraft, air traffic, aircraft navigation or aircraft communications; or in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

Sun Microelectronics

Contents



| | |
|--|-----------|
| 1. Overview | 1 |
| 1.1 Introduction | 1 |
| 1.2 U2S Executive Summary | 2 |
| 1.3 Design and Performance Goals | 3 |
| 1.4 Partition Overview | 4 |
| 1.5 U2S Overview / Block Diagram | 5 |
| 1.6 U2S Function Blocks Overview | 6 |
| 2. Pin Descriptions | 9 |
| 2.1 I/O Driver Specifications | 9 |
| 2.2 UPA Interface Signals | 10 |
| 2.3 SBus Interface Signals | 12 |
| 2.4 Total Pin Count | 14 |
| 3. Functional Description | 15 |
| 3.1 Functional Overview | 15 |
| 3.2 Top-Level Architectural Philosophy | 15 |
| 3.3 Internal Bus Block Diagrams | 16 |
| 3.4 Block Overviews | 19 |
| 4. Programming Model | 25 |
| 4.1 U2S Control/Status Register | 26 |
| 4.2 UPA Registers | 28 |
| 4.3 ECC Registers | 30 |
| 4.4 SBus Module | 34 |
| 4.5 IOMMU Registers | 41 |
| 4.6 Streaming Cache Registers | 51 |
| 4.7 Interrupt State Registers | 56 |
| 4.8 Counter/Timer Registers | 66 |
| 4.9 Performance Monitor Registers | 68 |

| | |
|--|------------|
| 5. PIO Decoder | 71 |
| 5.1 Definition of Terms | 71 |
| 5.2 Functional Description..... | 72 |
| 6. DMA Controller..... | 77 |
| 6.1 Definition of Terms | 77 |
| 6.2 Overview..... | 78 |
| 6.3 DMA Controller Functional Description | 84 |
| 7. Bus Controller | 87 |
| 7.1 Definition of Terms | 87 |
| 7.2 Overview..... | 88 |
| 7.3 Interface to Internal Blocks..... | 91 |
| 7.4 Bus Controller Functional Discription..... | 123 |
| 8. UPA Master / Slave Control..... | 131 |
| 8.1 Definition of Terms | 131 |
| 8.2 UPA Master / Slave Overview..... | 132 |
| 8.3 Signal Descriptions..... | 133 |
| 8.4 UPA Master / Slave Functional Description..... | 134 |
| 8.5 PIO Logic Descriptions | 137 |
| 9. UPA Reply Control..... | 139 |
| 9.1 Definition of Terms | 139 |
| 9.2 UPA Reply Overview..... | 140 |
| 9.3 Signal Descriptions..... | 142 |
| 9.4 UPA Reply Functional Description..... | 146 |
| 10. Ultra SPARC System ECC..... | 155 |
| 10.1 Definition of Terms | 155 |
| 10.2 Overview..... | 155 |
| 10.3 Signal Descriptions..... | 158 |
| 10.4 ECC Functional Description | 161 |
| 10.5 ECC Mondo Vector Unit Description..... | 164 |
| 10.6 PIO Logic Descriptions | 167 |
| 11. DMA Merge Buffer..... | 169 |
| 11.1 Definition of Terms | 169 |
| 11.2 DMA Merge Buffer Overview | 171 |
| 11.3 Functional Description..... | 172 |
| 12. SBus Module | 177 |
| 12.1 Overview..... | 177 |
| 12.2 SBus Module Functional Description | 181 |
| 12.3 Summary of Error Handling in SBM | 191 |

| | |
|--|------------|
| 13. SBus IOMMU | 193 |
| 13.1 Overview | 193 |
| 13.2 Mode of Operations | 194 |
| 13.3 Translation Storage Buffer | 196 |
| 13.4 Translation Errors..... | 200 |
| 13.5 IOMMU Demap..... | 200 |
| 13.6 TLB Initialization and Diagnostics..... | 200 |
| 14. Streaming Cache..... | 201 |
| 14.1 Overview | 201 |
| 14.2 Consistent DVMA and Stream DVMA | 201 |
| 14.3 Streaming Cache Management..... | 204 |
| 14.4 Streaming Cache Error Handling..... | 206 |
| 14.5 Software Notes..... | 206 |
| 15. Mondo Dispatch Unit..... | 207 |
| 15.1 Definition of Terms | 207 |
| 15.2 Overview | 208 |
| 15.3 Mondo Unit Functional Description..... | 211 |
| 15.4 Mondo Dispatch Timing Diagrams | 221 |
| 16. Timer Counter..... | 223 |
| 16.1 Overview | 223 |
| 16.2 Timer Signal Descriptions | 226 |
| 16.3 Timer Functional Descriptions | 227 |
| 17. Error Handling..... | 231 |
| 17.1 Overview | 231 |
| 17.2 Error Detection and Reporting | 231 |
| 17.3 Unreported Errors | 242 |
| 18. Scan / Jtag..... | 243 |
| 18.1 Introduction..... | 243 |
| 18.2 Features..... | 244 |
| 18.3 TAP signals..... | 244 |

U2S User's Manual

List of Figures



| | |
|---|-----|
| Typical Application Diagram | 4 |
| U2S Block Diagram | 5 |
| U2S Block Diagram | 16 |
| PIO Data and Address Paths | 17 |
| DMA Data and Address Paths | 18 |
| Breakdown of U2S Address Space | 26 |
| Interrupt Format Delivered to the Processor | 57 |
| Interrupt Format of Word 0 | 58 |
| PIO Transaction Flow | 72 |
| DMA Write to IO Space (Address only) | 79 |
| DMA Read Request to Memory (Address Only) | 80 |
| 64 Byte DMA Write Request to Memory | 81 |
| Less than 64 Byte DMA Write to Memory | 82 |
| Examples of Scoreboard Entries | 85 |
| PIO buses | 89 |
| DMA Buses | 90 |
| Generic Block Interface | 91 |
| PIO Read to an Internal Block Timing | 94 |
| PIO Write to an Internal Block Timing | 95 |
| Master Read Timing | 96 |
| Master Write Timing | 97 |
| IOMMU Interface | 100 |
| Mondo Dispatch Unit Interface | 101 |
| Merge Buffer Interface | 102 |
| SBus Module Interface | 104 |
| PIO Read to SBus Timing | 106 |
| PIO Write to SBus Timing | 106 |

| | |
|--|-----|
| SBus DMA Read Timing | 107 |
| SBus DMA Write Timing | 108 |
| Streaming Cache Interface | 110 |
| ECC/Parity Interface | 111 |
| Timer Counter Interface | 112 |
| Performance Monitor Interface | 112 |
| UPA Slave Receiver Interface | 113 |
| UPA Slave Receiver Signal Partition | 116 |
| UPA Slave Transmitter Interface | 117 |
| UPA Master Receiver Interface | 118 |
| UPA Master Transmitter Interface | 120 |
| DMA Controller Interface | 121 |
| UPA Master / Slave High-level block diagram | 132 |
| UM Block Diagram | 135 |
| UPA Slave Block Diagram | 136 |
| Block Diagram of PIO logic | 137 |
| Full Handshaking Circuit | 140 |
| UPA Reply High Level Block Diagram | 141 |
| FSM Block Diagram | 148 |
| Error Accumulation Logic | 149 |
| Data path flow from controller units to FIFO | 151 |
| P_REPLY FORMAT | 151 |
| P_REPLY Decoder | 152 |
| FSM Block Diagram | 152 |
| S_REPLY Block Diagram | 154 |
| Interface Block Diagram of ECC Unit | 156 |
| Block Diagram of ECC Unit | 157 |
| Check Block Diagram | 162 |
| ECC Generate Block Diagram | 164 |
| FSM Block Diagram | 165 |
| MV_RQ State Diagram | 166 |
| Block Diagram of PIO logic | 167 |
| Merge Buffer Overview | 171 |
| Merge Buffer Functional Block Diagram | 173 |
| Servicing a Partial Write | 175 |
| SBus Module Interface Diagram | 177 |
| SBM Block Diagram | 180 |
| SBus Arbiter Dead Cycle | 183 |
| PIO Dead Cycle | 185 |
| Address/Data Path Block Diagram | 189 |
| Virtual-to-Physical-Address Translation for 8K Page Size | 195 |

List of Figures

| | |
|---|-----|
| Virtual-to-Physical-Address Translation for 64K Page Size | 195 |
| Computation of TTE Entry Address | 198 |
| Mondo Dispatch Unit in U2S | 209 |
| Mondo Dispatch Overview Block Diagram | 210 |
| Mondo Vector Format on UPA Data Bus | 211 |
| Full INR Contents | 212 |
| Partial INR Contents | 213 |
| Interrupt Concentrator | 214 |
| Level Interrupt States | 215 |
| External Interrupt Concentrator Timing | 221 |
| Timer Block Diagram | 225 |
| TMR_FSM Block Diagram | 228 |
| TMR_FSM State Diagram | 229 |
| JTAG TAP Inputs and Outputs | 244 |

U2S User's Manual

Sun Microelectronics

x

List of Tables



| | |
|---|----|
| Best-Case Performance Goals | 3 |
| I/O Cells | 9 |
| UPA Signals | 10 |
| JTAG Signals | 12 |
| SBus Signals | 12 |
| Miscellaneous Signals | 13 |
| Total Pin Count | 14 |
| U2S Control Register Address | 26 |
| U2S Control Register Definition | 26 |
| Physical Address of UPA Registers | 28 |
| UPA Port ID Register | 28 |
| UPA Configuration Register | 29 |
| Physical Address of ECC Registers[] | 30 |
| ECC Control Register | 30 |
| ECC Error Reporting | 31 |
| UE AFAR | 32 |
| UE AFSR | 32 |
| CE AFSR | 33 |
| CE AFAR | 34 |
| Physical Address of SBus Registers | 34 |
| SBus Control Register | 35 |
| SBus AFSR | 39 |
| SBus AFAR | 39 |
| SBus Slot Configuration Register (per Slot) | 40 |
| Physical Address of IOMMU Registers | 41 |
| IOMMU Control Register | 42 |
| Address Space Size and Base Address Determination | 43 |

| | |
|--|----|
| IOMMU Translation Table Entry (TTE) | 44 |
| IOMMU Modes of Operation | 45 |
| TSB Base Address Register | 46 |
| Flush Address Register | 47 |
| TLB Tag Diagnostics Access | 48 |
| TLB Data RAM Diagnostics Access | 48 |
| LRU Entry Diagnostics Access | 49 |
| SBus Virtual Address Register | 50 |
| TLB Tag Comparator Diagnostics Access | 50 |
| Physical Address of Streaming Cache Registers | 51 |
| Streaming Cache General Control Register | 52 |
| Streaming Cache Page Invalidate/Flush Register | 52 |
| Streaming Cache Flush Synchronization Register | 53 |
| Streaming Cache Page Tag Format | 54 |
| Streaming Cache Line Tag Format | 54 |
| Streaming Cache Data RAM Content Format | 55 |
| Streaming Cache Data RAM Error Format | 55 |
| Physical Address of Interrupt Mapping Registers | 56 |
| INO Assignments | 59 |
| Physical Address of Clear Interrupt Pseudo Registers | 60 |
| Clear Interrupt Pseudo Register | 61 |
| Physical Address of Interrupt Retry Timer Registers | 62 |
| Interrupt Retry Timer Register | 62 |
| Physical Address of Interrupt State Diagnostic Registers | 63 |
| Interrupt State Meaning | 64 |
| SBus Internal Diagnostic Register Definition | 64 |
| OBIO and Miscellaneous Internal Diagnostic Register Definition | 64 |
| Physical Address of Counter/Timer Registers | 66 |
| Count Register | 66 |
| Limit Register | 67 |
| Physical Address of Performance Monitor Registers | 68 |
| Performance Monitor Control Register | 68 |
| Performance Counter Event Sources | 69 |
| Performance Counter Register | 70 |
| Size Encoding | 74 |
| Valid Slave Transactions | 74 |
| Slave Signals | 92 |
| Master Signals | 92 |
| Slave Buses | 93 |
| Master Buses | 93 |
| Block Prefixes | 99 |

List of Tables

| | |
|---|-----|
| Additional IOMMU Signals | 100 |
| Additional Mondo Dispatch Signals | 101 |
| Additional Merge Buffer Signals | 102 |
| Additional DMA Cache buses | 103 |
| Additional SBus Module Signals | 104 |
| SBM DVMA Consistent Transactions | 109 |
| Additional Streaming Cache Signals | 110 |
| Additional UPA Slave Receiver Signals | 114 |
| p_phdr | 117 |
| Additional UPA Master Receiver Signals | 118 |
| Destination/Source Encoding | 119 |
| UPA Interface Signals to DMA Controller Signals | 119 |
| Additional DMA Controller Signals | 121 |
| Additional DMA Controller buses | 122 |
| U2S Bus Controller Actions | 124 |
| S2U Bus Controller Actions | 126 |
| PIO Bus Controller Actions | 129 |
| Master/Slave Interface Signals | 133 |
| DMA Write Interface Signals | 142 |
| DMA Read Interface signals | 142 |
| PIO Write Interface Signals | 143 |
| PIO Read Interface Signals | 143 |
| Miscellaneous | 144 |
| P_REPLY signals | 145 |
| S_REPLY signals | 145 |
| U2S Requests - PIO Read Requests | 146 |
| U2S Requests- PIO Write requests | 146 |
| S2U Requests - DMA Read Requests | 147 |
| S2U Requests - DMA Write Requests | 147 |
| ECC Detect/Correct Signals | 158 |
| SEC/DED/S4ED Syndrome Encoding | 160 |
| Burst to non-Burst Examples | 184 |
| Summary of Error Handling in SBM | 191 |
| IOMMU Mode of Operation | 194 |
| TTE Data Format | 197 |
| Offset to TSB Table | 199 |
| Level Interrupt States | 215 |
| Interrupt Receiver State Register | 216 |
| Summary of Interrupts | 217 |
| External Interrupt Encoding | 221 |
| Timer Signals | 226 |

U2S User's Manual

| | |
|--|-----|
| Summary of Fatal Error Reporting in the Reference Platform | 239 |
| Summary of Non-Fatal Error Reporting in the Reference Platform | 239 |

1.1 Introduction

This manual describes the UltraSPARC™ Port Architecture (UPA) to SBus interface chip, also called the U2S^[1]. The U2S is the primary connection between the UPA port (including UltraSPARC-I processors and memory) and the I/O subsystem on an UltraSPARC-based CPU board. U2S features include:

1.1.1 Features

- Full master and slave port connection to the high-speed UPA interconnect. The UPA is a split address/data, packet-switched bus which runs up to 100 MHz in current implementations. UPA data is ECC protected
- Full master and slave SBus (IEEE P1496/Draft 2.3) with support for up to six SBus master devices (four slots + two onboard) and seven slave devices (four slots + three onboard). Logically the on board devices are identical to the SBus slots, but for electrical reasons, only four SBus expansion slots can be used
- 16-entry streaming cache for accelerating SBus Direct Virtual Memory Access (DVMA) activity
- 16-entry Input/Output Memory Management Unit (IOMMU) for mapping DVMA addresses

1. In previous documentation, the U2S was code named SYSIO.

- A “Mondo-Vector” Dispatch unit, or MDU, delivers interrupt requests in the format of packets to the UltraSPARC-I CPU modules
- Two SPARC V9 timer/counters with support for external cache
- Additional features to enhance the I/O subsystem

1.2 U2S Executive Summary

1.2.1 Pinout and Package

The U2S has 242 signal pins plus 65 V_{SS} and 65 V_{DD} pins for a total pin count of 372. See the Pin Description and Mechanical Information sections for a complete breakout of the signals.

1.2.2 Gate Count Estimate

The U2S uses 120K gates of random logic and 13,124 bits of SRAM.

1.2.3 Special Cell Requirements

Mixed-voltage operation: Internals and UPA I/O are 3.3 V, but SBus I/O is 5-V compatible. A 5-V reference pin is required.

- FIFOs and buffers use REGFILE cells. These are synchronous write/asynchronous read SRAMs
- Some SBus signals require holding amps
- CLK/CLK- require a pseudo-emitter-coupled-logic (PECL) input receiver and phase-locked loop (PLL) for this clock circuit. (Note that a 50-MHz clock domain does not need PLL)

1.2.4 Frequency of Operation

- The UPA operation is up to 100 MHz, SBus is at a fixed 25 MHz. Most of the internal logic runs at two times the SBus, or 50 MHz

1.2.5 Power Dissipation

- Power dissipation is less than 2.7 watts

1.3 Design and Performance Goals

The U2S is designed to optimize data transfers between the UPA and SBus. There are dedicated paths for programmed input/output (PIO) and DVMA requests and replies. This avoids unnecessary arbitration for internal paths to some degree.

The performance numbers in Table 1-1, “Best-Case Performance Goals,” are taken from simulations. For PIO and consistent DMA, ten back-to-back transactions were simulated. For streaming DMA, 128 back-to-back transactions were simulated.

Table 1-1 Best-Case Performance Goals

| Transfer Size (Bytes) | SBus Mode | PIO Write | PIO Read | DMA Write Consistent | DMA Read Consistent | DMA Write Streaming | DMA Read Streaming |
|-----------------------|-----------|-----------|----------|----------------------|---------------------|---------------------|--------------------|
| 1 | 32 bit | 8.62 | 3.01 | 2.66 | 1.49 | 4.96 | 4.84 |
| 2 | 32 bit | 17.24 | 6.02 | 5.32 | 2.98 | 9.83 | 9.45 |
| 4 | 32 bit | 34.48 | 12.05 | 10.75 | 5.95 | 19.31 | 18.03 |
| 8 | 32 bit | 51.28 | 22.22 | 20.62 | 11.30 | 31.41 | 29.36 |
| 16 | 32 bit | 67.80 | 35.40 | 39.60 | 20.62 | 45.76 | 44.10 |
| 32 | 32 bit | – | – | 67.23 | 36.53 | 66.58 | 61.13 |
| 64 | 32 bit | 89.39 | 66.12 | 80.40 | 55.36 | 79.94 | 72.44 |
| 8 | 64 bit | 68.97 | 24.39 | 20.83 | 11.90 | 30.84 | 28.04 |
| 16 | 64 bit | 83.33 | 43.01 | 40.00 | 22.86 | 50.10 | 45.51 |
| 32 | 64 bit | – | – | 76.92 | 41.24 | 88.73 | 70.77 |
| 64 | 64 bit | 148.15 | 98.77 | 124.03 | 76.19 | 122.93 | 106.06 |

Note: All bandwidth numbers are in MBytes/second.

Caution: Table 1-1 lists the performance goals for the U2S. These are the theoretical maximums the device is guaranteed not to exceed, not the minimum or typical performance. Many factors such can compromise performance. These factors include: system and SBus clock speeds; memory and SBus contention; different USC implementations; thrashing in the IOMMU and/or streaming cache due to large amounts of DVMA; response time of slow I/O devices (for example SBus PIO reads) that can’t be controlled; and software overhead.

1.4 Partition Overview

Figure 1-1, "Typical Application Design," illustrates the typical fit of the U2S into the system board. The U2S connects to the UPA address and data buses with the system controller (either the uniprocessor or multiprocessor version) providing UPA timing and control. On the SBus side, there are seven logical SBus slots, four of which are for expansion cards. The reset, interrupt, and clock controller (RIC) chip, provides the interrupts in an encoded format to save U2S pins.

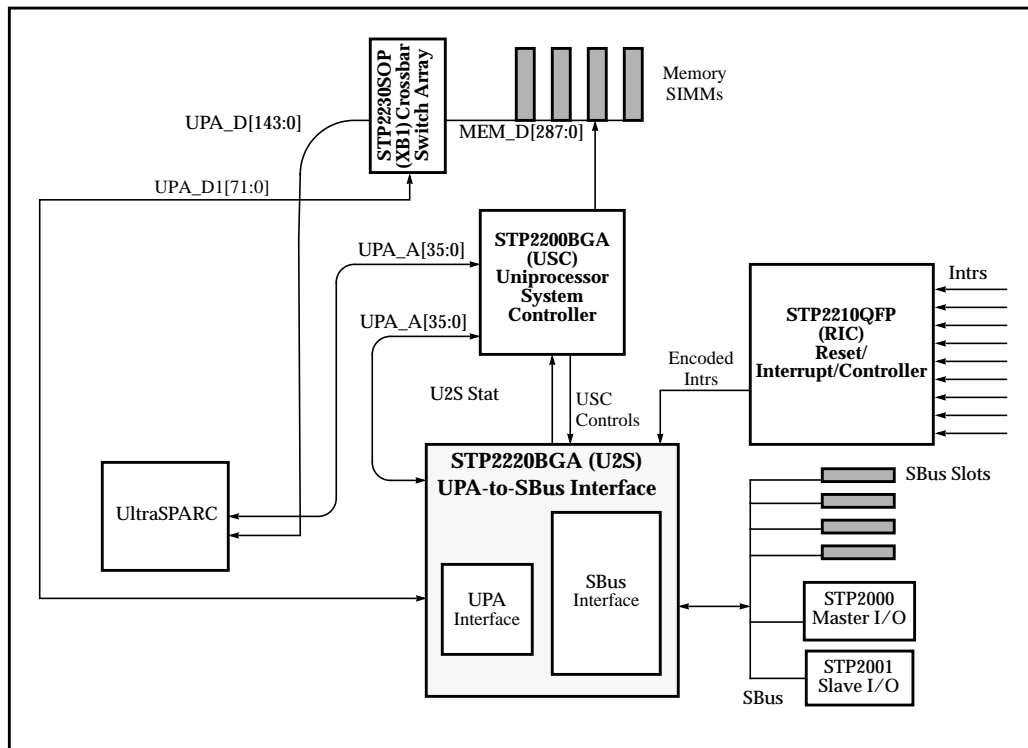


Figure 1-1 Typical Application Diagram

1.5 U2S Overview / Block Diagram

The U2S Block Diagram, Figure 1-2, shows the major symbolic blocks in the U2S and the general data/address flow. Section 1.6, “U2S Function Blocks Overview,” functionally describes these blocks.

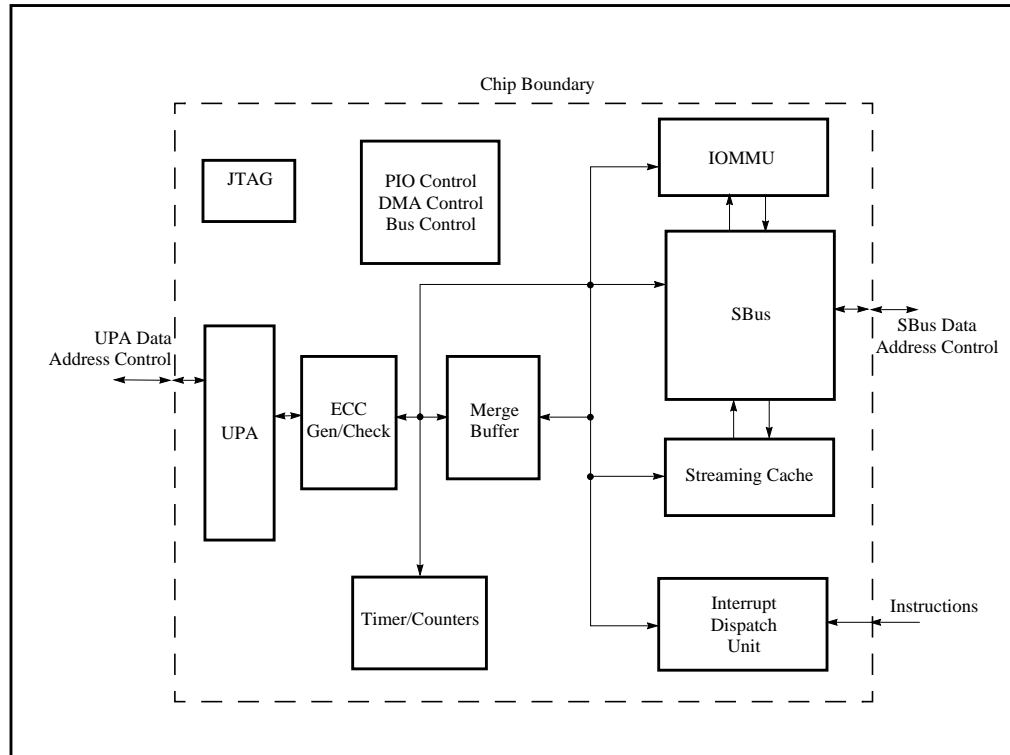


Figure 1-2 U2S Block Diagram

1.6 U2S Function Blocks Overview

This section briefly describes the U2S function blocks. For a more detailed description, refer to the "Functional Description" section or the individual sections which cover the detailed implementation. All function blocks in the U2S can be put into one of four categories:

- UPA
- SBus
- Mondo Interrupt
- Internal Control/Miscellaneous

1.6.1 UPA Interface Blocks

The UPA is a packet-switched main system bus. In an UltraSPARC-based system, the UPA can run up to 100 MHz, for example, a 10-ns clock cycle. Data and addresses have separate flow control. Each class of the UPA cycle uses its own FIFO-based queueing. In the U2S, this equates to a separate buffer for PIO read, PIO write, DMA read, and DMA write/Mondo. There is a synchronization boundary between the UPA blocks below and the other U2S blocks. The device runs at two times the SBus clock or 50 MHz. Control signals are two-clock synchronized between the domains.

- UMS (UPA_Master/Slave): The UMS block deals exclusively with UPA address control. It listens to the UPA_A when the U2S is a slave. It also arbitrates for and drives UPA_A when the U2S is the master. Data is a separate resource
- URP (UPA_Reply): The URP block deals exclusively with UPA data control. It generates the port reply (P_REPLY) to the USC during PIO cycles. It also listens to the system reply (S_REPLY) from the USC and manages the UPA data FIFOs
- Error Correcting Code (ECC) Generate: Generates ECC on a 64-bit data path
- ECC Check: Checks ECC on a 64-bit data path

1.6.2 SBus Interface Blocks

- SBM: The SBM is the main portion of the SBus interface. It performs all of the standard SBus master and slave protocol to the six logical masters and seven logical slaves which are supported in the U2S. The SBM controls SBus arbitration, flow control, and error handling
- IOMMU: IOMMU maps SBus virtual addresses to UPA physical addresses. It is used during DVMA only. The IOMMU caches recently-used translations in a translation lookaside buffer (TLB), and performs hardware table-walks
- SCache: The streaming cache (SCache) is used to accelerate SBus DVMA activity. For DVMA reads, the SCache will prefetch sequential 64-byte cache lines, and on DVMA writes, the SCache buffers 64-byte lines before writing them to memory. A tight coupling with the SBM module allows low latency for DVMA traffic that hits in the SCache

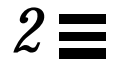
1.6.3 Mondo Interrupt

- MDU (Mondo Dispatch Unit): In SPARC V9 architecture, interrupts to the processor are sent as packets. The MDU block accepts interrupt requests from SBus, graphics, and other sources and dispatches interrupt packets to the UPA. These packets are treated similar to a DMA-write operation by the UPA interface

1.6.4 Internal Control

- Merge Buffer: In order to merge sub-line sized writes into a 64-byte memory line, it is necessary to perform a read-modify-write-type operation on the UPA. The Merge Buffer is responsible for generating the read, modifying the appropriate bytes, and writing the data back
- PIO Control: PIO Control decodes requests from the UPA_A request FIFO, arbitrates for the appropriate slot or internal U2S resource, and dispatches the request
- Bus Control: Bus Control is an internal arbiter shared between the DMA control and PIO Control units. It also schedules the use of internal resources such as MUXs
- DMA Control: DMA Control arbitrates and decodes requests from the SBM, SCache, IOMMU, and Mondo, and arbitrates for the appropriate UPA FIFO
- Timer Counters: There are two identical timer counters for use in system scheduling and profiling
- JTAG Control: The JTAG TAP controller controls boundary and internal scan functions. This implementation is compliant with IEEE Standard 1149.1

Pin Descriptions



This section describes the pinout of the U2S. Pin assignments are found in the section on Mechanical Information of the U2S Data Sheet.

2.1 I/O Driver Specifications

Table 2-1 describes the I/O cells used in the U2S.

Table 2-1 I/O Cells

| Driver Type | I/O | Drive (mA) | 5-V Tolerant | Description |
|-------------|-----|------------|--------------|---------------------------------------|
| B5IN03N | I | – | Y | LVTTL input |
| BIED03N | I | – | N | PECL input buffer (for clock) |
| BIN02N | I | – | N | LVTTL input |
| BIN06N | I | – | N | LVTTL input |
| PINA02N | I | – | N | LVTTL input w/ 50 K Ω pullup |
| PINX06N | I | – | N | LVTTL input w/ 50 K Ω pulldown |
| B5OZ10N | O | 8 | Y | Output |
| B5OZ20Y | O | 12 | Y | Output |
| BON10N | O | 10 | N | Output |
| BOZ24N | O | 24 | N | Output |
| BN02Z24N | I/O | 24 | N | Bidirectional no bus-hold |
| BN02Z24Y | I/O | 24 | N | Bidirectional no bus-hold |
| B5N3Z20Y | I/O | 12 | Y | Bidirectional no bus-hold |
| B5B3Z20L | I/O | 12 | Y | Bidirectional with bus-hold |
| B5B3Z20Y | I/O | 12 | Y | Bidirectional with bus-hold |

Note: Some of these buffers have identical functions although they have slightly different layouts on the silicon. This is true for the B5B3Z20L/B5B3Z20Y and the BN02Z24N/BN02Z24Y.

2.2 UPA Interface Signals

The signals in Table 2-2 connect from the U2S to the UPA.

Table 2-2 UPA Signals

| Signal Name | Pin Count | I/O | Driver | Description |
|------------------------------|-----------|-----|----------------------|-------------------------------------|
| UPA_DB[63:0] | 64 | I/O | BN02Z24Y or BN02Z24N | UPA 64-bit data bus |
| UPA_ECC[7:0] | 8 | I/O | BN02Z24N | UPA ECC bits for UPA data |
| ECC_VLD | 1 | I | BIN02N | UPA ECC valid |
| UPA_A[34:0] | 35 | I/O | BN02Z24N | UPA address |
| UPA_AP | 1 | I/O | BN02Z24N | UPA address parity (odd parity) |
| UPA_AV | 1 | I/O | BN02Z24N | UPA address valid |
| UPA_RIN[2:0] | 3 | I | BIN02N | UPA requests in |
| UPA_ROUT | 1 | O | BOZ24N | UPA request out asserted by the U2S |
| UPA_CR | 1 | I | BIN02N | UPA request from the USC |
| UPA_ARBRST | 1 | I | BIN02N | UPA arbiter reset |
| UPA_PRLY[4:0] | 5 | O | BOZ24N | UPA P_REPLY (port reply) |
| UPA_SRLY[4:0] | 5 | I | BIN02N | UPA S_REPLY (system reply) |
| UPA_DTST | 1 | I | BIN02N | Data stall |
| UPA_RST | 1 | I | PINX02N | UPA reset |
| UPA_CLK | 1 | I | BIED03N | UPA system clock high (PECL) |
| $\overline{\text{UPA_CLK}}$ | 1 | I | BIED03N | UPA system clock low (PECL) |
| UPA Total | 130 | | | |

2. Pin Descriptions

The signals in Table 2-3 are used for Scan/JTAG.

Table 2-3 JTAG Signals

| Signal Name | Pin Count | I/O | Driver | Description |
|-------------------------------|-----------|-----|---------|------------------|
| SIO_TMS | 1 | I | PINA02N | Test mode select |
| SIO_TDO | 1 | O | B5OZ10N | Scan data out |
| SIO_TDI | 1 | I | PINA02N | Scan data in |
| $\overline{\text{SIO_TRST}}$ | 1 | I | PINA02N | Test reset |
| SIO_TCLK | 1 | I | BIN02N | Test clock |
| JTAG Total | 5 | | | |

2.3 SBus Interface Signals

The signals in Table 2-4 connect from the U2S to the SBus.

Table 2-4 SBus Signals

| Signal Name | Pin Count | I/O | Driver | Description |
|----------------------------------|-----------|-----|----------------------------|---|
| SB_PA[27:0] | 28 | I/O | B5B3Z20Y or B5B3Z20L | SBus address bus |
| SB_D[31:0] | 32 | I/O | B5B3Z20Y or B5B3Z20L | SBus data bus |
| SB_DP | 1 | I/O | B5B3Z20Y | Parity bit for the data bus |
| SB_SIZ[2:0] | 3 | I/O | B5N3Z20Y | Size of transaction |
| SB_RD | 1 | I/O | B5N3Z20Y | Read / $\overline{\text{Write}}$ signal |
| $\overline{\text{SB_AS}}$ | 1 | O | B5OZ20Y | Address strobe |
| SB_ACK[2:0] | 3 | I/O | B5N3Z20Y | Acknowledge signals |
| $\overline{\text{SB_LERR}}$ | 1 | I/O | B5N3Z20Y | Late error |
| SB_SEL[6:0] | 7 | O | B5OZ20Y | Slave slot selects |
| SB_CLK | 1 | I | B5IN03Y | SBus clock (input, for reference) |
| $\overline{\text{SB_BR}}$ [5:0] | 6 | I | B5IN03Y | SBus request lines |

Table 2-4 SBus Signals

| Signal Name | Pin Count | I/O | Driver | Description |
|---------------------------------|-----------|-----|---------|------------------|
| $\overline{\text{SB_BG}}[5:0]$ | 6 | O | B5OZ20Y | SBus grant lines |
| $\overline{\text{SB_RST}}$ | 1 | O | B5OZ20Y | SBus reset |
| Total SBus | 91 | | | |

The miscellaneous signals are shown in Table 2-5.

Table 2-5 Miscellaneous Signals

| Signal Name | Pin Count | I/O | Driver | Description |
|--------------------------------|-----------|-----|---------|-------------------------------------|
| TMR_CLK | 1 | I | B5IN03N | Timer counter clock |
| INT_NUM[5:0] | 6 | I | B5IN06Y | Interrupt number |
| SYSIO_CLK | 1 | I | B5IN06Y | 50-MHz clock input |
| $\overline{\text{SLV_RST}}$ | 1 | I | B5IN06Y | STP2001 chip reset |
| REF_5V | 1 | I | – | 5-V reference for mixed-voltage I/O |
| $\overline{\text{BIST_MODE}}$ | 1 | I | PINA02N | BIST test mode enable |
| PM_OUT | 1 | O | BON10N | Process monitor output |
| PLL_BYPASS | 1 | I | BIN06N | Bypass the PLL on UPA_CLK |
| UPA_CLK_OUT | 1 | O | BOZ24N | UPA_CLK out for lab debug only |
| Spare | 2 | – | – | Unused pins |
| Total | 16 | | | |

2.4 Total Pin Count

The U2S package is a 372-pin BGA. The total pin count by function is shown in Table 2-6.

Table 2-6 Total Pin Count

| Interface | Pin Count |
|-----------------|-----------|
| UPA | 130 |
| JTAG/Scan | 5 |
| SBus | 91 |
| Miscellaneous | 16 |
| Power/Ground | 130 |
| Total pin count | 372 |

3.1 Functional Overview

This chapter concerns the functional description of the U2S device at the top level. Overall device design is discussed and the address/data flow is presented. The following implementation sections contain in-depth descriptions of each sub-block. The chapter has two major sections:

- Block diagrams of the address and data paths
- A description of each of the major design blocks within the U2S

The first few sections of the implementation section (PIO decode, DMA Controller, and Bus Controller) cover examples of typical PIO and DMA cycles.

3.2 Top-Level Architectural Philosophy

When reviewing the U2S internal bus structure, note the architectural philosophy of the device:

- There is an UPA
- There is an SBus
- The UPA and SBus are being integrated so that data goes as fast as possible between the two busses without making the device too complicated

3.3 Internal Bus Block Diagrams

The U2S Block Diagram illustrated in Chapter 1, "Overview" is repeated in Figure 3-1, "U2S Block Diagram."

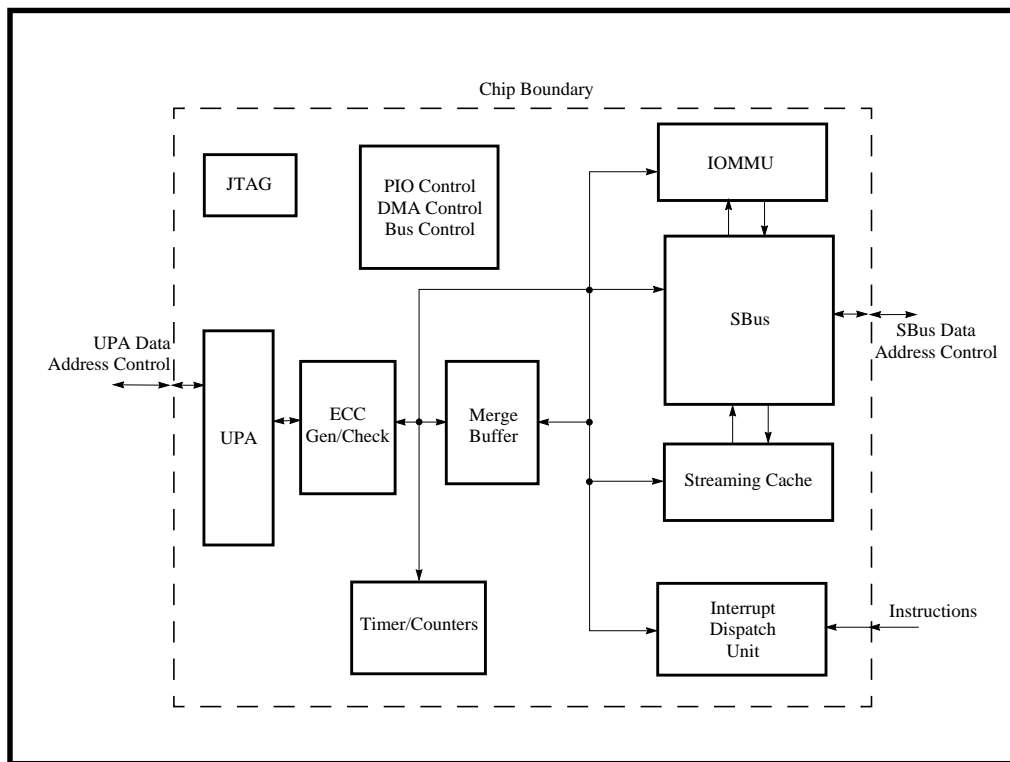


Figure 3-1 U2S Block Diagram

3. Functional Description

Figure 3-2, "PIO Data and Address Paths," shows the PIO data and address paths.

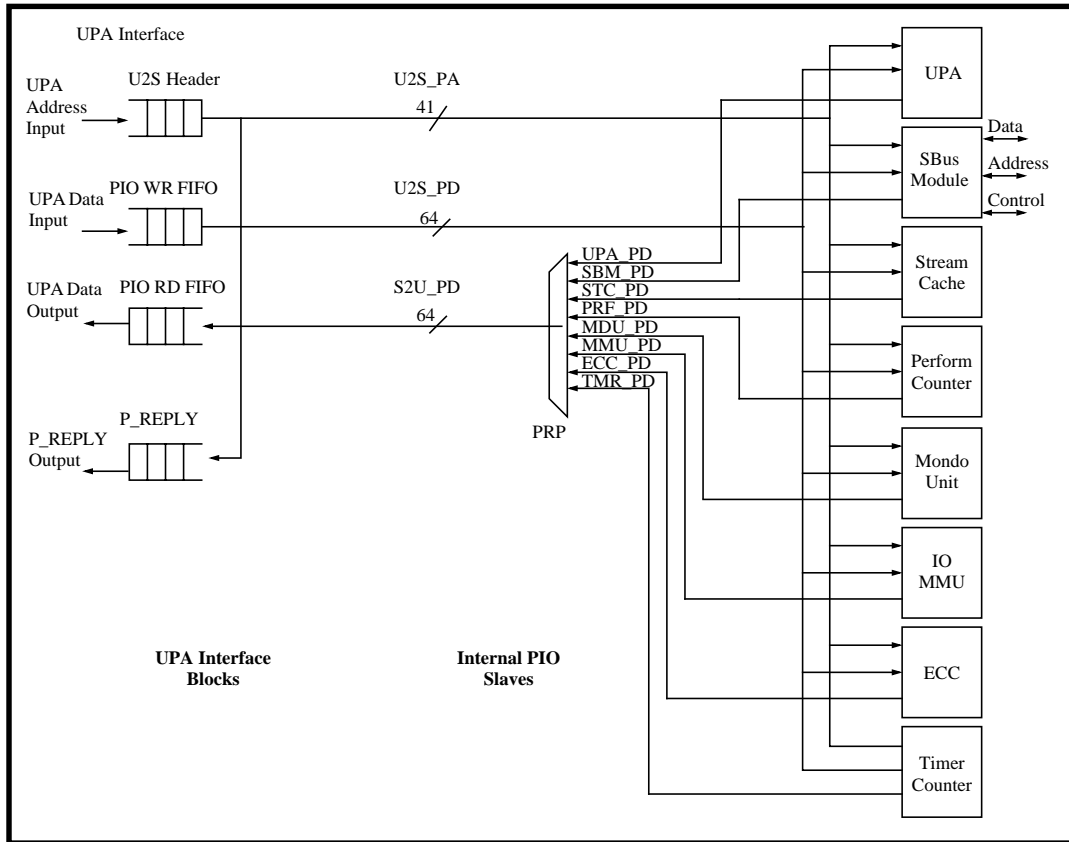


Figure 3-2 PIO Data and Address Paths

Figure 3-3, "DMA Data and Address Paths," illustrates the DMA data and address paths.

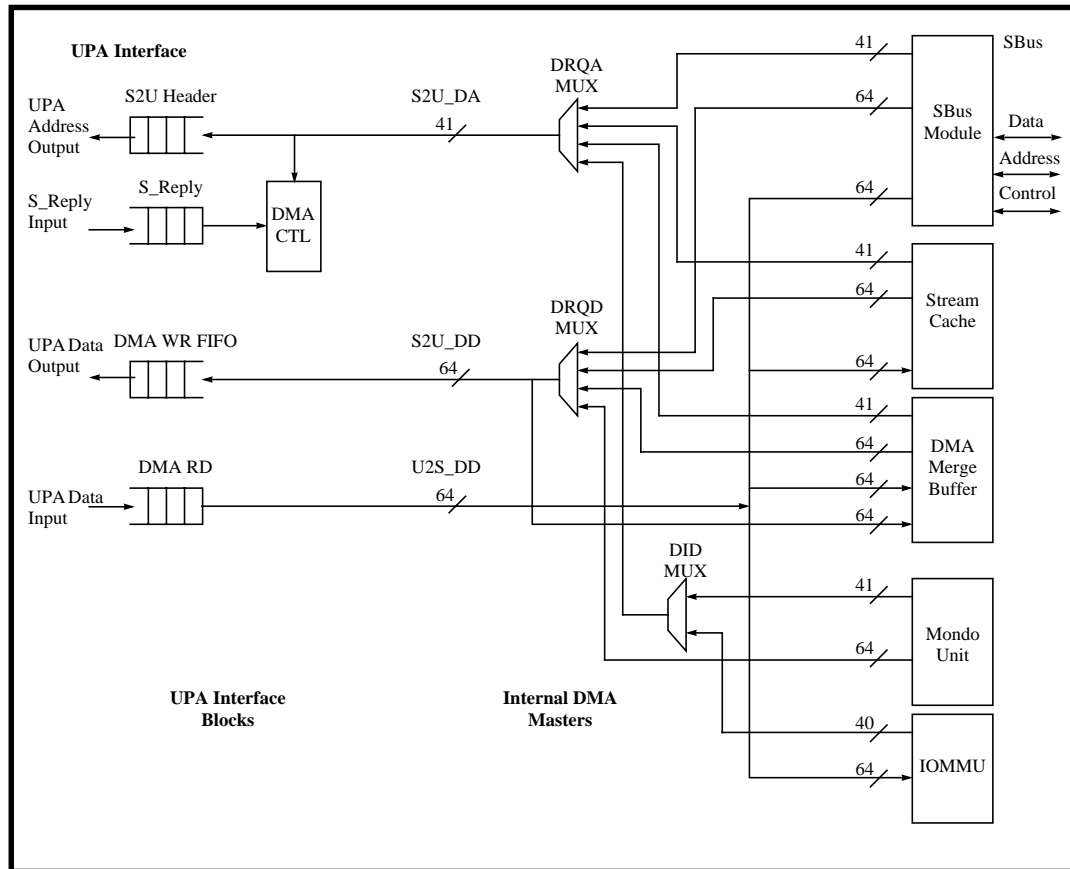


Figure 3-3 DMA Data and Address Paths

3.4 Block Overviews

The following sections describe the U2S control sections.

3.4.1 PIO Decoder

This block decodes requests from the U2S header request FIFO. The target of the PIO cycle may be either the SBus or one of the internal U2S units. For SBus, the SBM unit will determine the transaction timing based on the response of the SBus device. For internal units, the PIO-control and bus-control units control the timing. All PIO requests are serviced strictly in order and one at a time.

3.4.2 DMA Control

This block decodes and builds the UPA packet for requests from the SBus, Streaming Cache, IOMMU, Mondo Dispatch Unit, and Merge Buffer. This block also keeps a FIFO of the requests. The UPA will service U2S DMA requests in order.

3.4.3 Bus Control

This is an internal arbiter shared between the DMA-control and PIO-control units. There are not really any “busses” in the U2S, i.e., there are no tristate signals. The bus-control unit handles timing of internal U2S resources based on the number of clocks required for each kind of transaction.

3.4.4 UPA Master/Slave (UMS)

The UMS block is the U2S's interface to the UPA_A request bus. When the U2S is addressed as an UPA slave (PIO requests), the UMS always writes the request into the U2S header FIFO.

When the U2S is performing a DMA request, the UMS arbitrates for use of the UPA_A bus and drives the request out from the S2U header FIFO.

As with the UPA reply block, the UMS runs at the UPA clock frequency, up to 100 MHz. Signals from other internal U2S blocks are two-clock synchronized by the UMS before being used. Likewise, status and control, which is an output of the UMS block, needs to be two-clock synchronized with two times the SBus clock (50 MHz) before being used.

3.4.5 UPA Reply

The UPA reply unit manages replies to the USC device. P_Reply requests are received from the PIO-control unit and forwarded to the USC. For PIO reads, P_Reply indicates to the USC that the U2S has read data ready in the PIO_RD FIFO. For PIO writes, P_Reply indicates that the U2S has removed write data from the PIO_WR FIFO. P_Reply is not used for DMA.

S_Reply is used for both PIO and DMA cycles. For PIO reads, S_Reply indicates that the USC is ready to read data out of the PIO_RD FIFO. For PIO writes, S_Reply indicates that the USC is writing data into the PIO_WR FIFO. For DMA reads, S_Reply indicates that the USC is delivering data into the DMA_RD FIFO, and for DMA writes, S_Reply indicates that the USC is ready to read data out of the DMA_WR FIFO.

The UPA reply unit is responsible for reading and writing the appropriate data FIFOs and enabling the UPA data outputs if necessary. In addition, the UPA reply unit must control the ECC check logic. When data is arriving at the U2S (related to a PIO write or DMA read), the data is ECC checked. The UPA reply unit must accumulate the ECC results for the entire packet, which may be between 1 and 64 bytes in length. The UPA reply unit manages a separate packet-status FIFO which signals the PIO- and DMA-control units of error conditions.

3.4.6 ECC Generate/Check

The ECC unit is split into separate generate and check functions. ECC is always calculated on 64 bits of data. The ECC generate logic is positioned on the “internal” side of all of the data FIFOs. (There is not enough time between receiving S_Reply and providing data to generate the ECC if the logic is on the UPA side of the FIFO.) This may also allow more flexibility in the circuit timing since the UPA clock is faster than the internal U2S clock. In some situations, it is necessary for the U2S to generate intentionally bad ECC with the data. Two of the ECC check bits are inverted to provide a guaranteed uncorrectable error.

The ECC check logic will detect correctable (CE) and uncorrectable (UE) ECC errors. Refer to the ECC section for the conditions for detecting and correcting errors. For a correctable error, the data will be repaired before being sent to the internal destination block. None of the internal U2S units will be aware of CE errors. For both UE and CE errors, the ECC unit will generate a Mondo vector interrupt to the processor.

3.4.7 DMA Merge Buffer

The DMA Merge Buffer block is used for servicing DMA writes of less than 64 bytes (partial writes). This is required in an UltraSPARC based system because there is no way to write cacheable memory in sub-line increments. (There are at least two problems that force caching: the SIMMS do not have individual byte controls and ECC is generated on 8-byte boundaries.) In order to perform a sub-line write, the U2S must perform a UPA Read-To-Own (RTO) transaction to gain control of the line, merge the new data into the line, and then flush the line to memory.

To avoid the complexity of having to participate in system coherence, during a merge the USC blocks all requests to the line in which U2S has issued an RTO until the data merge is completed and the U2S has issued a write-back of the line to memory.

The DMA Merge Buffer is not intended to be high speed for sub-line DMA writes from the SBus, rather its purpose is to provide correct functionality to the UPA. The SCache is used to improve SBus performance by buffering data into 64-byte lines before flushing to memory. SCache line flushes that contain a complete 64 bytes will be able to bypass the merge buffer by doing a write invalidate on the UPA.

3.4.8 SBM

The SBus module block implements a complete SBus master and slave interface. The SBM implements all of the required bus controller functions for SBus: `SB_BR` arbitration, SBus reset, etc. The SBM also implements the SBus time-out counter. The SBM module runs at two times the SBus clock (50 MHz) and samples the SBus clock to know when to generate SBus control signals. There is a synchronization boundary between the SBM and the UPA interface. The SBM supports the IEEE P1496 specification, including the 64-bit extended-transfer modes. This allows transfer sizes from 1 to 64 bytes. Data can be transferred either 32 or 64 bits at a time.

SBM handles the timing of PIO requests to the SBus. These are handled one at a time. The SBM implements dynamic sizing, retries, and standard SBus functionality.

DVMA cycles are more complicated. The SBM communicates with the IOMMU and SCache modules to complete DVMA cycles on the SBus. DVMA data can be moved from the SBus to either (a) another SBus device, (b) the SCache, or (c) the UPA. Based on the IOMMU mapping information for the virtual address, the SBM will treat the DVMA cycle as either a *consistent* or *streaming* access.

Consistent accesses are sent directly to the SBus and have strict ordering constraints. The performance of consistent accesses is also much slower than streaming accesses and DVMA pages should only be marked consistent-mode when necessary. The SBM must also guarantee ordering between DVMA-write cycles and Mondo vectors. There are interlocks between the SBM and MDU unit to prevent an interrupt from “passing” a data transaction that occurred first.

The only SBus function that is not handled by SBM is the interrupt logic. This is contained in the MDU unit.

3.4.9 IOMMU

The IOMMU block is used for SBus DVMA cycles. It maps 32-bit SBus virtual addresses to 41-bit UPA physical addresses. There is a 16-entry TLB to cache recently-used translations. The IOMMU can provide two levels of service when the SBus presents a virtual address for translation:

- First, the IOMMU searches the TLB to see if the VA -> PA translation is already available
- If there is a miss on the TLB, the IOMMU block will perform a hardware table-walk to get the translation. The IOMMU does this by reading from the Translation Storage Buffer (TSB) table by issuing a DMA read to main memory. This is only a single-level table search

3.4.10 SCache

The Streaming Cache is used to accelerate DVMA traffic from the SBus. It contains a pool of 16×64 -byte entries. These entries are tagged by virtual page number, and are managed as a fully-associative cache. Only one entry will be valid for any given virtual page. Entries are assigned as needed by the SCache logic. An LRU algorithm is used to assign new pages to entries when all of the entries are valid. All 16 entries are available for read or write streams.

For DVMA writes, the SCache will buffer up data in an entry until 64 bytes have been filled. The SCache will then flush the completed line to memory. For DVMA reads, the SCache will try to prefetch new 64-byte lines into an entry before they are needed by the requesting device. Note that the first access in each page will cause an SCache miss, subsequent accesses in the same page will be prefetched. The SCache expects that the DVMA device is accessing data in sequential and increasing order. If the actual device access pattern is different the SCache will maintain data correctness but the performance gains will not be as high.

Data which is stored in the SCache does *NOT* participate in the UPA coherence algorithm. The SCache implements a flush command to allow system software to explicitly remove virtual pages from the SCache when a DVMA transfer is done or when an IOMMU damp operation occurs. In order to ensure that the SCache flush data has reached UPA memory a special synchronization register is implemented.

3.4.11 Mondo Dispatch Unit

The Mondo Dispatch Unit (MDU) is the U2S's vehicle for dispatching interrupt packets to the UltraSPARC-I CPUs in an UltraSPARC-based system. The MDU will generate a special type of UPA packet (a Mondo vector) with fields indicating the interrupt number of the interrupt. This block accepts interrupt requests from SBus, graphics, and internal U2S sources and dispatches interrupt packets to the UPA. The contents and target of the interrupt packet are controlled through the Interrupt Number Registers (INR) within the MDU. Each INR is 16 bits, with five of these bits indicating the Master ID (MID) of the target CPU. For simplicity, no valid data is sent in the interrupt packet from the U2S.

There are approximately 37 external interrupt sources that can have their requests serviced through the MDU. In order to save U2S pins, an *interrupt concentrator* has been included within the RIC. The interrupt concentrator sends interrupt requests to the U2S using a simple handshake. There are six lines to send the encoded interrupt level number and two lines for the handshake.

Once the interrupts are received from the RIC they are broken into two groups based on the LSB of the MID target field. An optimized UltraSPARC-based system has no more than two CPUs. Within each of these groups, the MDU performs a priority arbitration to determine which interrupt to send to each CPU. The Mondo interrupt packet is then sent to the target CPU using the same UPA queue that is used for DMA writes. The Mondo interrupt packet can be either ACKed or NACKed by the UPA. If the packet is NACKed (rejected), the CPU is already busy servicing another Mondo. In this case, the MDU will resubmit the packet at a later time based on a free-running programmable retry interval counter. In the meantime, Mondo packets can still be sent to other CPUs. To simplify the design, the U2S will only have one Mondo vector outstanding (for example, waiting for ACK or NACK) at any time.

3.4.12 *Timer/Counters*

There are two independent identical timers in the U2S. Each provides either periodic interrupts or alarm-clock (callout) interrupts to a selected processor. The interrupt is delivered to the target CPU using a Mondo vector.

3.4.13 *Performance Counters*

There are two performance counters in the U2S which monitor certain events in the U2S. The events to monitor are programmable (see UltraSPARC reference platform specification) and include the following: Number of streaming DVMA reads, number of streaming DVMA writes, number of consistent DVMA reads, number of consistent DVMA writes, number of TLB misses, number of streaming buffer misses, number of cycles SBus is granted to DVMA, number of bytes transferred using DVMA, number of interrupts, number of interrupt NACKs, number of PIO reads, number of PIO writes, number of SBus reruns, and number of U2S cycles SBus is consumed by PIO.

Each of the two counters can monitor independent events and are simply incremented each time the event occurs. The counters are cleared and can be changed to monitor any event dynamically by software.

3.4.14 *JTAG*

The U2S will have JTAG scan capability on both boundary and internal flip-flops. More details of this capability are found in Chapter 18, "Scan/Jtag."

Register accesses to the U2S can be in any size from one byte up to eight bytes. Sizes and locations for the registers are given in the sections which follow. Reads of any size up to eight bytes to any register are supported regardless of whether reads of that size make sense. Writes of any size up to eight bytes are also supported regardless of whether writes of that size make sense. Writes of any size *may* corrupt unwritten bits in the register (for example, writes may result in all eight bytes being written regardless of the indicated write size). Software must ensure that only the proper sized accesses are used. No hardware checking is performed. Burst access to U2S registers is not permitted and will result in an error return for reads, and silent failure for writes. Unpredictable results may also result from misaligned access, as occurs if the “E” bit isn’t set correctly in the Translation Table Entry (TTE).

Figure 4-1 shows the rough breakdown of addresses within the U2S. The upper four GB are used for pass through, while the lower four GB address internal registers to the U2S.

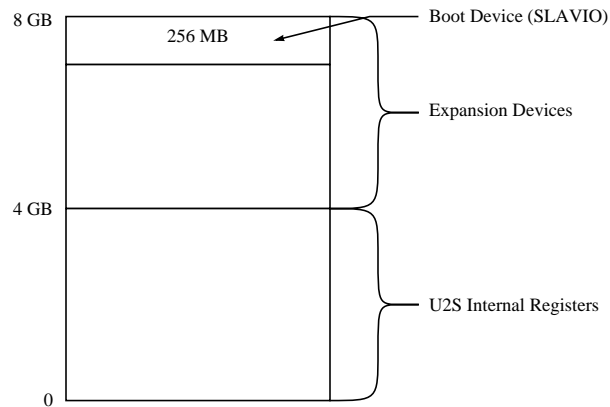


Figure 4-1 Breakdown of U2S Address Space

4.1 U2S Control/Status Register

Table 4-1 U2S Control Register Address

| Register | Physical Address | Access Size |
|----------------------|------------------|-------------|
| U2S control register | 0x1FE.0000.0010 | 8 bytes |

Table 4-2 U2S Control Register Definition

| Field | Bits | Description | Type |
|-------|-------|---|------|
| IMPL | 63:60 | Implementation number of U2S (0) | R |
| VER | 59:56 | Revision number of this implementation (0) | R |
| MID | 55:51 | UPA MID for U2S. Bits [55:51] contain 0x1F on reset. Software should set up MID before allowing U2S to generate interrupt or DVMA. Refer to the Port ID listed in Table 4-4 | R/W |
| IGN | 50:46 | Interrupt Group Number; sets the five bits of the IGN field supplied in the first word of an interrupt packet. Refer to "Interrupt Mapping Registers" on page 56 | R/W |

Table 4-2 U2S Control Register Definition

| Field | Bits | Description | Type |
|------------|------|---|-------|
| Reserved | 45:5 | | R0 |
| CLK_OUT_EN | 4 | Clock output enable. Defaults to one (the clock output is enabled). Should be set to 0 by initialization of software. Useful for checking the behavior of the internal PLLs | R0 |
| APCKEN | 3 | Address parity check enable. Defaults to 0 at power up. When set, any parity error detected results in a P_FERR reply. If clear, parity errors are still logged (in APERR), but the transaction continues as though the parity is correct | R/W |
| APERR | 2 | Incoming system address parity error. Persistent across reset. Is set regardless of the value in APCKEN | R/W1C |
| IAP | 1 | Invert UPA address parity. Reset to 0. The U2S generates odd parity when this bit is set to 0 and even parity when set to 1 | R/W |
| MODE | 0 | Specify the speed of the U2S clock relative to the UPA clock. MODE = 1 if the UPA clock is faster than the U2S clock, and 0 if equal to or slower. Set to 0 upon reset | R/W |

The design of the U2S is optimized with the assumption that the UPA is running faster than the U2S clock. This assumption is true during normal operation of the system. However, in the debug or bring-up stage of the system, this assumption may not be true. The MODE bit allows the UPA to run slower than the U2S clock, which is fixed at twice the speed of the SBus clock. Operations proceed normally regardless of the state of this bit and are guaranteed to work with MODE set to 0. Configuration software should set the MODE bit to one if it determines that the UPA clock is faster than the U2S clock, which is fixed at 50 MHz in normal operation.

4.2 UPA Registers

Table 4-3 Physical Address of UPA Registers

| Register | Physical Address | Access Size |
|----------------------------|------------------|-------------|
| UPA port ID register | 0x1FE.0000.0000 | 8 bytes |
| UPA configuration register | 0x1FE.0000.0008 | 8 bytes |

4.2.1 UPA Port ID Register

This register includes information about identification and capability of the U2S UPA interface. This register is read-only.

Table 4-4 UPA Port ID Register

| Field | Bits | Description | Type |
|--------------|-------|---|------|
| FCODE_ESCAPE | 63:56 | Value is 0xFC, which is the FCODE for FERR. Software attempts to read this register as FCODE will be readily identifiable as errors | R |
| Reserved | 55:35 | Reserved, read as 0 | R |
| ECC NotValid | 34 | Indicates whether this port can generate ECC when sourcing data. Set to 0 | R |
| ONEREAD | 33 | Set if the slave port can allow only one outstanding slave read P_REQ transaction at a time. This bit is set to 0 | R |
| Reserved | 32:31 | Reserved, read as 0. Would encode PINT_RDQ for ports implementing this feature | R |

Table 4-4 UPA Port ID Register

| Field | Bits | Description | Type |
|---------|-------|--|------|
| PREQ_DQ | 30:25 | Specify the size of data queue in 16-byte units. This field is 0x8 for the U2S which has 128 bytes of data queue | R |
| PREQ_RQ | 24:21 | Specify the size of the PREQ_RQ queue. The U2S can support two pending PREQ, this field is 0x2 | R |
| UPACAP | 20:16 | Bit [16]: set if the port has master capability; set to 1. Bit [17]: set if the port has a cache; set to 0. Bit [18]: set if the port uses UPA_SLAVE_INT signal; set to 0. Bit [19]: set if the port can generate interrupt; set to 1. Bit [20]: set if the port can service interrupt; set to 0 | R |
| JEDEC | 15:00 | JEDEC identification (0xEF07) | R |

4.2.2 UPA Configuration Register

This register indicates the queue sizes for each class of UPA request. Please refer to the system controller (SC) for the description of classes. The U2S only uses one request class for its transfer. The depth of queue supported by the USC for the U2S is two, therefore the SCIQ0 field should be programmed with 0x2. The initial value after reset is 0x1.

Table 4-5 UPA Configuration Register

| Field | Bits | Description | Type |
|----------|------|--|------|
| Reserved | 63:8 | Reserved, read as 0, and write has no effect. | R |
| SCIQ1 | 7:4 | Unused, read as 0, and write has no effect. | R0 |
| SCIQ0 | 3:0 | Size of input request queue for one master class in the USC. Software should set it to 0x2 during initialization. ^[1] | R/W |

1. Software must ensure that NO DMA is taking place when this field is changed. Once set, the value may not be reduced; it can only be increased.

4.3 ECC Registers

For information on which of the interrupt vectors these errors are reporting see Table 4-40, "INO Assignments."

Table 4-6 Physical Address of ECC Registers^[1]

| Register | Physical Address | Access Size |
|----------------------|------------------|-------------|
| ECC control register | 0x1FE.0000.0020 | 8 bytes |
| Reserved | 0x1FE.0000.0028 | 8 bytes |
| UE AFSR | 0x1FE.0000.0030 | 8 bytes |
| UE AFAR | 0x1FE.0000.0038 | 8 bytes |
| CE AFSR | 0x1FE.0000.0040 | 8 bytes |
| CE AFAR | 0x1FE.0000.0048 | 8 bytes |

1. Note: UE, CE, AFSR, and AFAR will be defined in the following tables.

4.3.1 ECC Control Register

This register controls enable/disable of ECC checking and generation of ECC error-related interrupts. All bits are 0 upon reset.

Table 4-7 ECC Control Register

| Field | Bits | Description | Type |
|----------|------|--|------|
| ECC_EN | 63 | Enables ECC checking. ECC generation is always enabled. | R/W |
| UE_INTEN | 62 | Enable interrupt generation on uncorrectable error (UE). | R/W |
| CE_INTEN | 61 | Enables interrupt on correctable ECC errors (CE). | R/W |

Table 4-8, “ECC Error Reporting,” shows how the ECC_EN and UE_INTEN/CE_INTEN controls the ECC checking and error handling in the U2S.

Table 4-8 ECC Error Reporting

| ECC_EN | INTEN | Description |
|--------|-------|--|
| 0 | X | No ECC checking and reporting, every UPA transaction proceeds as if there is no ECC error. Data flows through from UPA to SBus. |
| 1 | 0 | ECC checking will be done, but no interrupt will be sent on ECC error. UE on PIO write will not be performed on SBus, UE on DVMA read will return SBus error acknowledgment. Error is logged in AFSR/AFAR but no interrupt is generated. Software should clear error status before enabling interrupt. |
| 1 | 1 | U2S send interrupt on ECC error. UE on PIO write will not be performed on SBus, UE on DVMA read will return SBus error acknowledgment. The error is logged in the AFSR/AFAR. |

4.3.2 Uncorrectable Error Asynchronous Fault Status/Address Register (UE AFSR/AFAR)

Any uncorrectable ECC error detected by the UPA interface of the U2S will log the error in the UE AFSR/AFAR. Uncorrectable errors can happen during PIO write, DVMA read, or DVMA partial write. Two sets of status bits are defined in this register. Bits [63:61] are the primary error status and bits [60:58] are the secondary status. Only one of the primary error status can be set at any time. Primary error status can be set only when:

- none of the primary error conditions exists prior to this error, or
- the new error detected at the same time software is clearing the primary error; the same time means on coincident clock cycles. Setting takes precedence over clearing

Secondary bits are set whenever a primary bit is set (only one primary bit can be set at a time). The secondary bits are cumulative and always indicate that information has been lost as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits [47:37] of AFSR log address and status of the primary UE. Further UE will not be logged into these bits until software clears the primary error, which makes the AFAR and part of the AFSR available to log new error. An interrupt is generated whenever the AFAR logs the new error address, if the interrupt is enabled.

Table 4-9 UE AFSR

| Field | Bits | Description | Type |
|-----------|-------|--|-------|
| P_PIO | 63 | Set if primary UE is caused by PIO access. | R/W1C |
| P_DRD | 62 | Set if primary UE is caused by SBus DVMA read. | R/W1C |
| P_DWR | 61 | Set if primary UE is caused by SBus DVMA write. | R/W1C |
| S_PIO | 60 | Set if secondary UE is caused by PIO access. | R/W1C |
| S_DRD | 59 | Set if secondary UE is caused by SBus DVMA read. | R/W1C |
| S_DWR | 58 | Set if secondary UE is caused by SBus DVMA write. | R/W1C |
| Reserved | 57:48 | Reserved, read as 0 | R |
| DW_OFFSET | 47:45 | Offset of double-word containing ECC error in 64-byte block, relative to PA modulo 64 bytes. | R |
| SIZE | 44:42 | Size of failed primary transfer is 2^{SIZE} bytes. ^[1] | R |
| UPA_MID | 41:37 | UPA MID that caused the error transaction. | R |
| Reserved | 36:00 | Reserved, read as 0 | R |

1. In the event that byte merging occurs at the processor (the E bit is incorrectly set), such that the byte enables are not meaningful, the size returned in this field is unpredictable. The size may not reflect the actual DVMA transfer size on the SBus if the primary error is caused by a DVMA access.

Table 4-10 UE AFAR

| Field | Bits | Description | Type |
|----------|-------|---------------------------------------|------|
| Reserved | 63:41 | Reserved, read as 0 | R |
| UE_PA | 40:00 | Physical address of error transaction | R |

4.3.3 Correctable Error Asynchronous Fault Status/Address Register (CE AFSR/AFAR)

The U2S logs the correctable ECC error in the CE AFSR/AFAR. Correctable errors can happen during PIO write, DVMA read, or DVMA partial write. Two sets of status bits are defined in this register. Bits [63:61] are the primary error status and bits [60:58] are the secondary error status. Only one of the primary error status can be set at any time. Primary error status can be set only when

- none of the primary error condition exists prior to this error, or
- a new error detected at the same time software is clearing the primary error; the same time means on coincident clock cycles. Setting takes precedence over clearing

Secondary bits are set whenever a primary bit is set (only one primary bit can be set at a time). The secondary bits are cumulative and always indicate that information has been lost as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits [55:37] of the AFSR log address and status of the primary CE. Further CE will not be logged into these bits until software clears the primary error, which makes the AFAR and part of the AFSR available to log a new error. An interrupt is generated whenever the AFAR logs the new error address, if the interrupt is enabled.

Table 4-11 CE AFSR

| Field | Bits | Description | Type |
|----------|-------|--|-------|
| P_PIO | 63 | Set if primary CE is caused by PIO access | R/W1C |
| P_DRD | 62 | Set if primary CE is caused by SBus DVMA read | R/W1C |
| P_DWR | 61 | Set if primary CE is caused by SBus DVMA write | R/W1C |
| S_PIO | 60 | Set if secondary CE is caused by PIO access | R/W1C |
| S_DRD | 59 | Set if secondary CE is caused by SBus DVMA read | R/W1C |
| S_DWR | 58 | Set if secondary CE is caused by SBus DVMA write | R/W1C |
| Reserved | 57:56 | Reserved, read as 0 | R |
| E_SYND | 55:48 | CE syndrome bits | R |

Table 4-11 CE AFSR

| Field | Bits | Description | Type |
|---------------------|-------|---|------|
| DW_OFFSET | 47:45 | Offset of double-word containing ECC error in 64-byte block, relative to PA modulo 64 bytes | R |
| SIZE ^[1] | 44:42 | Size of failed primary transfer is 2 ^{SIZE} bytes | R |
| UPA_MID | 41:37 | UPA MID that caused the error transaction | R |
| Reserved | 36:00 | Reserved, read as 0 | R |

1. In the event that byte merging occurs at the processor (the E bit is incorrectly set), such that the byte enables are not meaningful, the size returned in this field is unpredictable. The size may not reflect the actual DVMA transfer size on the SBus if the primary error is caused by a DVMA access.

Table 4-12 CE AFAR

| Field | Bits | Description | Type |
|----------|-------|---------------------------------------|------|
| Reserved | 63:41 | Reserved, read as 0 | R |
| UE_PA | 40:00 | Physical address of error transaction | R |

4.4 SBus Module

The SBus Module contains the registers which control SBus operations. One SBus Slot Configuration Register (SSCR) exists per slot and contains information specific to a SBus slot. The SBus Control Register contains information that is common to all SBus slots supported by the U2S. All slots, except slot 15, support master capability.

Table 4-13 Physical Address of SBus Registers

| Register | Physical Address | Access Size |
|------------------------------------|------------------|-------------|
| SBus control register | 0x1FE.0000.2000 | 8 bytes |
| Reserved | 0x1FE.0000.2008 | 8 bytes |
| SBus AFSR | 0x1FE.0000.2010 | 8 bytes |
| SBus AFAR | 0x1FE.0000.2018 | 8 bytes |
| SBus Slot 0 configuration register | 0x1FE.0000.2020 | 8 bytes |
| SBus Slot 1 configuration register | 0x1FE.0000.2028 | 8 bytes |

Table 4-13 Physical Address of SBus Registers

| Register | Physical Address | Access Size |
|--|------------------|-------------|
| SBus Slot 2 configuration register | 0x1FE.0000.2030 | 8 bytes |
| SBus Slot 3 configuration register | 0x1FE.0000.2038 | 8 bytes |
| SBus Slot 13 configuration register ^[1] | 0x1FE.0000.2040 | 8 bytes |
| SBus Slot 14 configuration register ^[1] | 0x1FE.0000.2048 | 8 bytes |
| SBus Slot 15 configuration register ^[1] | 0x1FE.0000.2050 | 8 bytes |

1. While listed as general SBus slots, these are strictly reserved for on board IO devices. The assignment is implementation specific. On the UltraSPARC system, the ordering is as follows: slot 13 is audio (APC), slot 14 is either Macio or FEPS, and slot 15 is Slavio.

4.4.1 SBus Control Register

Table 4-14 SBus Control Register

| Field | Bits | Description | Type |
|---------------|-------|--|-------|
| IMPL | 63:60 | Implementation number of host adapter. This field is hardwired to 0x0 | R |
| REV | 59:56 | Revision number for this design; initially 0 | R |
| Reserved | 55:54 | Read as 0 | R |
| DMA_PERR[5:0] | 53:48 | Set when DVMA write parity error is detected from SBus slot. Bits [3:0] correspond to SBus slots 3 through 0, respectively. Bit 5 corresponds to slot 14 and bit 4 to slot 13 | R/W1C |
| Reserved | 47 | Read as 0 | R |
| PIO_PERR[6:0] | 46:40 | Set when PIO load parity error occurs. These errors result in bad ECC delivered to the processor, so system software should check here to determine whether the bad ECC comes from a data path error or a logical (PIO load parity). See description below | R/W1C |
| Reserved | 39:11 | Reserved. Read as 0 | R |
| FAST_SBUS | 10 | Used to shorten the PIO access latency. See the description below. Reset to 0 | R/W |

Table 4-14 SBus Control Register

| Field | Bits | Description | Type |
|-------------|------|---|------|
| WAKEUP_EN | 9 | Power management wakeup enable control; 1 for power management wakeup enabled, 0 for disabled. Reset to 0 | R/W |
| ERRINT_EN | 8 | Enable SBus error interrupt, reset to 0. This is only for errors reported in the SBus AFSR | R/W |
| Reserved | 7:6 | Reserved | R |
| ARB_EN[5:0] | 5:0 | SBus DVMA arbitration enable. Reset to 0x0. See the description below for bit assignments | R/W |

IMPL

IMPL is the implementation number of this host adapter for the UltraSPARC-I reference platform. This is the first implementation of an UltraSPARC adapter.

REV

REV is the revision number for the design.

DMA_PERR[5:0]

DMA_PERR bits are associated with SBus master requests, both on-board and expansion. They are set whenever the U2S detects a DVMA write parity error from an associated master. These bits are write-1-to-clear. The bit assignments are listed below.

- DMA_PERR[3:0]: SBus slot 3-0
- DMA_PERR[4]: APC
- DMA_PERR[5]: MACIO

Note: DMA writes to the slot configuration registers which result in a parity error do not set the DMA_PERR for that slot. Instead, $\overline{\text{SB_LERR}}$ will be generated to the master, and no logging takes place. Writes with bad parity, whether to memory or UPA devices, result in a bad ECC being written to the target and are logged with these bits.

PIO_PERR[6:0]

PIO_PERR is set when a PIO load parity error occurs. These errors result in a bad ECC delivered to the processor, so system software should check here to determine whether the bad ECC comes from a data path error or a logical PIO load parity. These bits are write-1-to-clear. The bit assignments are listed below.

- PIO_PERR[3:0]: SBus slot 3-0
- PIO_PERR[4]: APC
- PIO_PERR[5]: MACIO
- PIO_PERR[6]: SLAVIO

FAST_SBUS

The FAST_SBUS bit controls the access latency for PIOs to the SBus. To expedite the transfer, the acknowledgment to the UPA (P_REPLY) is delivered to the UPA before the data is available. Normally, data will be available before the S_REPLY is delivered by the USC to the U2S. This bit should be set to one except for the cases where the UPA clock frequency is slower than the U2S clock frequency. This bit has been included for debug purposes. So long as the SBus clock is running at 25 MHz, this bit can be set to one, even with a maximum-rate (10 ns cycle time) UPA clock.

WAKEUP_EN

WAKEUP_EN is used by system software when putting various parts of the system to sleep. When set to 1, any attempt by an enabled SBus device (ones with ARB_EN set to 1) to arbitrate for the SBus will result in an interrupt packet being delivered to a target for the purpose of waking up the system. See the section concerning interrupts for more details. Arbitration for SBus devices is inhibited as long as this bit is set: in other words, with WAKEUP_EN set to one, the system behaves as though ARB_EN[5:0] is set to zero.

Additionally, this bit enables a divide by 1000 prescaler for the system timer/counters thereby changing them from incrementing once per microsecond to once per millisecond.

ERRINT_EN

ERRINT_EN enables interrupt generation on SBus errors when set to one. Interrupts are disabled when set to 0. Power-up state is zero.

ARB_EN

ARB_EN enables DVMA arbitration for SBus slots 3-0, audio (APC), and the MACIO/SLAVIO. If set to zero, the bus request from the slot will be ignored by the SBus arbiter. Assignment of enables is as shown below:

0-3 => SBUS slots 0 through 3

4 => APC (audio)

5 => MACIO/SLAVIO

4.4.2 SBus Asynchronous Fault Status/Address Registers

The SBus AFSR/AFAR record error information related to PIO read/write to SBus slave devices. Only asynchronous errors reported through interrupt are recorded in these registers. Asynchronous errors include errors triggered by any PIO write to SBus devices, PIO read, and SBus late error.

Two sets of status bits are defined in this register. Bits [63:61] are the primary error status and bits [60:58] are the secondary error status. Only one of the primary error status can be set at any time. Primary error status can be set only when

- none of the primary error conditions exist prior to this error, or
- a new error is detected at the same time software is clearing the primary error

Secondary bits are set whenever a primary bit is set (only one primary bit can be set at a time). The secondary bits are cumulative and always indicate that information has been lost as no address information has been captured. Setting of the primary error bits is independent.

The AFAR and bits [47:37] of AFSR log address and status of the primary SBus PIO error. Further SBus PIO errors will not be logged into these bits until software clears the primary error, which makes the AFAR and part of the AFSR available to log a new error. An interrupt is generated whenever the AFAR logs the new error address. Bits below S_BERR have no meaning if no primary or secondary error is set.

4. Programming Model

Table 4-15 SBus AFSR

| Field | Bits | Description | Type |
|----------|-------|--|-------|
| P_LE | 63 | Set if the primary error detected is a late PIO error | R/W1C |
| P_TO | 62 | Set if the primary error detected is a SBus time-out (PIO wait) | R/W1C |
| P_BERR | 61 | Set if the primary error detected is a SBus error ack (PIO wait) | R/W1C |
| S_LE | 60 | Set if the secondary error detected is a late PIO error | R/W1C |
| S_TO | 59 | Set if secondary error detected is a SBus time-out (PIO wait) | R/W1C |
| S_BERR | 58 | Set if the secondary error detected is a SBus error ack (PIO wait) | R/W1C |
| Reserved | 57:48 | Reserved, read as 0 | R |
| RD | 47 | 1 = primary error is caused by PIO late read 0 = primary error is caused by PIO write | R |
| Reserved | 46:45 | Reserved, read as 0 | R |
| SIZE | 44:42 | Size of failed primary transfer is 2^{SIZE} bytes. | R |
| MID | 41:37 | UPA MID that caused error transaction | R |
| Reserved | 36:00 | Reserved, read as 0 | R |

Table 4-16 SBus AFAR

| Field | Bits | Description | Type |
|----------|-------|---------------------------------------|------|
| Reserved | 63:41 | Reserved, read as 0 | R |
| PA | 40:00 | Physical address of error transaction | R |

4.4.3 SBus Slot Configuration Register (SSCR)

All information in the SSCR is accessible by PIO read/write. Part of the information is accessible by the corresponding slot DVMA master through SBus virtual address 0x4000.0000 when a “BY” bit in the register is set. The “BY” bit can be set only by the host processor. Bits [31:00] are readable by the slot DVMA master. Only bits [26:15] are writable from slot DVMA master. All bits in this register are reset to 0.

Table 4-17 SBus Slot Configuration Register (per Slot)

| Field | Bits | Description | Type |
|-------------|-------|--|------|
| Reserved | 63:27 | Reserved bits, read as 0 | R |
| SEGA[40:30] | 26:16 | Segment address; provides PA[40:30] when bypass mode is used | R/W |
| CP | 15 | Bypass mode access is cacheable | R/W |
| ETM | 14 | This slot slave interface supports extended transfer mode | R/W |
| PE | 13 | Enable SBus parity checking; generation is always enabled | R/W |
| Reserved | 12:5 | Reserved, read as 0 | R |
| BA64 | 4 | This slot slave interface supports 64-byte burst transfers | R/W |
| BA32 | 3 | This slot slave interface supports 32-byte burst transfers | R/W |
| BA16 | 2 | This slot slave interface supports 16-byte burst transfers | R/W |
| BA8 | 1 | This slot slave interface supports 8-byte burst transfers | R/W |
| BY | 0 | IOMMU bypass mode enabled | R/W |

Caution: Do not set BA8 through BA64 if a device is incapable of responding to that size access. For example, the SLAVIO will not operate correctly if any of these bits are set.

4.5 IOMMU Registers

Table 4-18 Physical Address of IOMMU Registers

| Register | Physical Address | Access Size |
|--|--------------------------------------|-------------|
| IOMMU control register | 0x1FE.0000.2400 | 8 bytes |
| TSB base address register | 0x1FE.0000.2408 | 8 bytes |
| IOMMU flush register | 0x1FE.0000.2410 | 8 bytes |
| SBus virtual address diagnostic register | 0x1FE.0000.4400 | 8 bytes |
| TLB tag compare diagnostics | 0x1FE.0000.4408 | 8 bytes |
| IOMMU LRU queue diagnostics | 0x1FE.0000.4500 – 0x1FE.0000.457F | 8 bytes |
| TLB tag diagnostics | 0x1FE.0000.4580 – 0x1FE.0000.45FF | 8 bytes |
| TLB data RAM diagnostics | 0x1FE.0000.4600 – 0x1FE.0000.46FF | 8 bytes |

4.5.1 IOMMU Control Register

The IOMMU Control Register provides means to enable and disable the diagnostic mode, translation storage buffer (TSB) size, and page size. It also contains some revision-control information.

Note: This register physically resides in the SBus module, so it will not provide synchronization with any other IOMMU register. Similarly, on writes to this register, the software will need to either read this register or another in the SBus module to guarantee write completion.

Table 4-19 IOMMU Control Register

| Field | Bits | Description | Type |
|-------------------------|-------|---|------|
| IMPL | 63:60 | IOMMU implementation number, hardwired to 0 | R |
| REV | 59:56 | IOMMU revision number of current implementation: 0 for first revision | R |
| Reserved | 55:19 | Reserved, read as 0 | R |
| TSB_SIZE | 18:16 | TSB table size measured in the number of 8-byte entries: 0=1K, 1=2K, 2=4K, 3=8K, 4=16K, 5=32K, 6=64K, 7=128K | R/W |
| Reserved | 15:3 | Reserved, read as 0 | R |
| TBW_SIZE ^[1] | 2 | Assumed page size during TSB lookup: 0 = 8K page 1 = 64K page | R/W |
| MMU_DE | 1 | Diagnostic mode enable, when set it enables the diagnostic mode. See description of TLB tag diagnostics | R/W |
| MMU_EN | 0 | IOMMU enable bit, when set, it enables the translation | R/W |

1. If DVMA mappings are always 8K pages, or mixed 8K and 64K pages, set this bit to 0 so that the index is constructed for 8K lookup. If all DVMA mappings are to 64K pages, set this bit to 1 so that the index is based on 64K pages. When this bit is 0 a 64K mapping should be placed in all eight TSB entries in which it is indexed.

4. Programming Model

Address space size and base address are controlled by TSB_SIZE and TBW_SIZE as shown in Table 4-20. For reference, the TTE format has been included.

Table 4-20 Address Space Size and Base Address Determination

| TSB_SIZE | TBW_SIZE == 0 | | | TBW_SIZE == 1 | | |
|----------|---------------|--------------------------------|-----------|---------------|--------------------------------|-----------|
| | VA Space Size | VA Base Address ^[1] | TSB_Index | VA Space Size | VA Base Address ^[1] | TSB_Index |
| 0 | 8 MB | 0xFF80.0000 | VA[22:13] | 64 MB | 0xFC00.0000 | VA[25:16] |
| 1 | 16 MB | 0xFF00.0000 | VA[23:13] | 128 MB | 0xF800.0000 | VA[26:16] |
| 2 | 32 MB | 0xFE00.0000 | VA[24:13] | 256 MB | 0xF0000000 | VA[27:16] |
| 3 | 64 MB | 0xFC00.0000 | VA[25:13] | 512 MB | 0xE000.0000 | VA[28:16] |
| 4 | 128 MB | 0xF800.0000 | VA[26:13] | 1 GB | 0xC000.0000 | VA[29:16] |
| 5 | 256 MB | 0xF000.0000 | VA[27:13] | 2 GB | 0x8000.0000 | VA[30:16] |
| 6 | 512 MB | 0xE000.0000 | VA[28:13] | 4 GB | 0x0000.0000 | VA[31:16] |
| 7 | 1 GB | 0xC000.0000 | VA[29:13] | Not allowed | – | – |

1. Software should use the value shown. It has been chosen to clearly eliminate aliasing. No hardware checks are performed to prevent aliasing.

4.5.2 IOMMU Translation Table Entry (TTE)

Table 4-21 IOMMU Translation Table Entry (TTE)

| Bits | Field | Use |
|----------|-----------------|--|
| [63] | DATA_V | Valid: Indicates that this TTE is valid |
| [62] | Reserved | Read as 0 |
| [61] | DATA_SIZE | Page Size: 0 == 8 KB, 1 == 64 KB |
| [60] | STREAM | If set, the page is streamable |
| [59] | LOCAL_BUS | If set, the access is to the same bus segment |
| [58:51] | DATA_SOFT_2 | Assigned for software use |
| [50:N] | Reserved | Room for growth, variable physical address size |
| [N-1:13] | DATA_PA[N-1:13] | Physical page number |
| [12:7] | DATA_SOFT | Assigned for software use |
| [6:5] | Reserved | Read as 0 |
| [4] | CACHEABLE | 1 == access to cacheable space 0 == access to noncacheable space |
| [3:2] | Reserved | Read as 0 |
| [1] | DATA_W | Writable page. Attempts to write to a page with this bit 0 will result in an error ack |
| [0] | Reserved | Read as 0 |

While software should replace “x” with “0” for legal codes, hardware will interpret combinations of Stream, Local_Bus, and cacheable as follows:

Table 4-22

| Stream | Local_Bus | Cacheable | Meaning |
|--------|-----------|-----------|---|
| x | 1 | x | Access to slave on same I/O bus segment |
| 0 | 0 | 0 | Non-cacheable access to system |
| 0 | 0 | 1 | Consistent-mode cacheable access to system memory |
| 1 | 0 | x | Streaming mode cacheable access to system memory |

4. Programming Model

The physical address of DVMA transaction is generated based on the value of the MMU_EN of the U2S control register, “BY” bit of the SBus slot configuration register, and SBus virtual address (VA) bits [31:30]. Table 4-23 shows the various modes of IOMMU.

Table 4-23 IOMMU Modes of Operation

| MMU_EN | BY | VA[31:30] | Mode |
|--------|----|-----------|------------------------------------|
| X | 1 | 00 | Bypass |
| X | 1 | 01 | Slot configuration register access |
| 1 | 1 | 1X | Translation |
| 0 | 1 | 1X | Pass-through |
| 0 | 0 | XX | Pass-through |
| 1 | 0 | XX | Translation |

In the bypass mode, the physical address is formed by concatenating SEGA[40:30] of the SBus slot configuration register with SBus virtual addresses [29:00]. In pass-through mode, the physical address is formed by appending 9 or 10 bits of zero onto bits 31:0 or 30:0 of the SBus VA. In pass-through mode, accesses are only to memory space and are performed only in consistent mode. The access is considered cacheable, and coherent operation will be generated to UPA. In translation mode, the physical address is obtained by performing a VA-to-PA translation through TLB. The SBus slot configuration register can be accessed directly by the SBus master through VA[31:30] = 01, while the bypass enable bit, BY, is set to one.

4.5.3 TSB Base Address Register

The TSB Base Address Register contains the pointer to the first-entry of the TSB table. Together with part of the virtual address, it uniquely identifies the address where hardware should fetch the TTE from the TSB table. The TSB table has to be aligned on an 8 K boundary. The lower order 13 bits are assumed to be 0x0 during TSB table lookup. Tables larger than 8 Kbytes are only constrained to be on 8 K boundaries rather than having to be size aligned.

Table 4-24 TSB Base Address Register

| Field | Bits | Description | Type |
|----------|-------|---|------|
| Reserved | 63:41 | Reserved, read as 0 | R |
| TSB_BASE | 40:13 | Upper 28 bits of the SBus TSBs physical address | R/W |
| Reserved | 12:0 | Reserved, read as 0 | R |

4.5.4 Flush Address Register

The Flush Address Register is a write-only pseudo-register to allow software to perform an address-based flush of a mapping from the TLB. The data written to this address contains the page number to be flushed. A TLB entry with matched page number will be invalidated.

Table 4-25 Flush Address Register

| Field | Bits | Description | Type |
|-----------|-------|---|------|
| Reserved | 63:32 | Reserved, write has no effect | W |
| FLUSH_VPN | 31:13 | 31:16 = virtual page number if 64K page; bits 15:13 are don't care 31:13 = virtual page number if 8K page | W |
| Reserved | 12:0 | Reserved, write has no effect | W |

On any cycle, an entry may be looked up in the TLB either by the flush hardware or the DVMA hardware. Simultaneous flush and use cannot occur. Devices attempting to use an entry which software flushes will either get the entry before the flush if the request beats the flush, or will find the entry missing from the TLB and will perform a hardware table-walk. Software should invalidate the entry from the TSB before issuing the flush.

Note: Completion of any read to the IOMMU (except the IOMMU control register) within the U2S guarantees that the flush transaction has completed.

4.5.5 TLB Tag Diagnostics Access

The TLB Tag Diagnostics Access provides a diagnostics path to the 16-entry TLB tag when the MMU_DE bit in the IOMMU control register is turned on.

Table 4-26 TLB Tag Diagnostics Access

| Field | Bits | Description | Type |
|----------|-------|---|------|
| Reserved | 63:22 | Reserved, read as 0 | R |
| W | 21 | Writable bit, when set, the page mapped by the TLB has write permission granted | R/W |
| S | 20 | Stream bit, 1 = page is streamable, 0 = page is not streamable | R/W |
| SIZE | 19 | Page size, 0 = 8 K and 1 = 64 K | R/W |
| VPN | 18:0 | VPN[31:13] | R/W |

Note: Diagnostic accesses should ensure that multiple match conditions are not generated. The result of multiple matches is unpredictable.

4.5.6 TLB Data RAM Diagnostic Access

The TLB Data RAM Diagnostics Access provides direct PIO accesses to 16 entries of TLB data RAM. The MMU_DE bit in the IOMMU control register must be turned on to perform the accesses. Table 4-27 shows the information included in the returned data.

Table 4-27 TLB Data RAM Diagnostics Access

| Field | Bits | Description | Type |
|----------|-------|---|------|
| Reserved | 63:31 | Reserved, read as 0 | R |
| V | 30 | Valid bit, when set, the TLB data field is meaningful | R/W |
| L | 29 | Local_Bus to indicate the direction of DVMA transfer. 1 = access local to the U2S (intra SBus-to-SBus transfer), 0 = non-local access (DVMA to memory or the UPA) | R/W |

Table 4-27 TLB Data RAM Diagnostics Access

| Field | Bits | Description | Type |
|-----------|------|---|------|
| C | 28 | Cacheable bit. 1 = Cacheable access, 0 = Non-cacheable Access type is strictly determined by this bit, not by PA[40] | R/W |
| PA[40:13] | 27:0 | 28-bit physical page number | R/W |

4.5.7 LRU Queue Diagnostic Access

The LRU Queue Diagnostic Access can be directly accessed by PIO read for diagnostic purposes. The MMU_DE bit in the IOMMU control register must be set to perform direct access. There are 16 entries in the LRU queue. Each entry contains a unique value range from 0x0 to 0xF. Entry 0 contains the pointer to a TLB entry which is least recently used, and entry 15 contains the pointer to a TLB entry that is most recently used.

Table 4-28 LRU Entry Diagnostics Access

| Field | Bits | Description | Type |
|----------|------|---------------------|------|
| Reserved | 63:4 | Reserved, read as 0 | R |
| LRU_DO | 3:0 | LRU entry selected | R |

4.5.8 SBus Virtual Address Diagnostic Register

This register is used to set up the virtual address for the TLB compare diagnostic. The virtual address is written to this register and the compare results from TLB can be read.

Table 4-29 SBus Virtual Address Register

| Field | Bits | Description | Type |
|----------|-------|--------------------------|------|
| Reserved | 63:32 | Reserved, read as 0 | R |
| SBUS_VPN | 31:13 | SBus virtual page number | R/W |
| Reserved | 12:00 | Reserved, read as 0 | R |

4.5.9 TLB Tag Compare Diagnostic Access

Table 4-30 TLB Tag Comparator Diagnostics Access

| Field | Bits | Description | Type |
|----------|-------|---|------|
| Reserved | 63:16 | Reserved, read as zeros | R |
| COMP | 15:0 | TLB tag comparator output for each entry. | R |

Note: The TLB Tag Comparator Diagnostics Access provides a diagnostics path to the 16-entry TLB Tag Comparator when the MMU_DE bit in the IOMMU Control Register is turned on. Bit 0 represents the comparison result of the first TLB Tag entry, and bit 15 represents the last.

In order to avoid invalid address translation after TLB diagnostics, the valid bits in the TLB should be reset appropriately before doing any meaningful address translation. Diagnostics write to read-only space or read from write-only space will be ignored.

4.6 Streaming Cache Registers

This section describes the registers associated with the streaming cache. This block inside the U2S is used for speeding up DMA transfers which are less than 64 bytes in size by providing read-ahead and write-behind buffering of data. Streaming DMA enters the coherence domain at a different time than consistent transfers with reads leaving the coherent space earlier and writes entering the coherent space later.

Note: While the streaming cache is a separate block inside the U2S, its Streaming Cache Control Register logically resides in the SBus module. This means that an access designed to assure write completion should either access a streaming cache register or one in the SBus module.

Table 4-31 Physical Address of Streaming Cache Registers

| Register | Physical Address | Access Size |
|--|--------------------------------------|-------------|
| Streaming Cache control register | 0x1FE.0000.2800 | 8 bytes |
| Streaming Cache page flush/invalidate register | 0x1FE.0000.2808 | 8 bytes |
| Streaming Cache flush synchronization register | 0x1FE.0000.2810 | 8 bytes |
| Streaming Cache data RAM diagnostic | 0x1FE.0000.5000 – 0x1FE.0000.53FF | 8 bytes |
| Streaming Cache error status diagnostics | 0x1FE.0000.5400 – 0x1FE.0000.57FF | 8 bytes |
| Streaming Cache page tag diagnostics | 0x1FE.0000.5800 – 0x1FE.0000.587F | 8 bytes |
| Streaming Cache line tag diagnostics | 0x1FE.0000.5900 – 0x1FE.0000.597F | 8 bytes |

The streaming cache performs all operations on 8 KB pages. Transfers larger than 8 KB, even though they may be mapped with a single IOMMU entry, require eight flush operations.

4.6.1 Streaming Cache Control Register

This register controls the various functions of the streaming cache.

Table 4-32 Streaming Cache General Control Register

| Field | Bits | Description | Type |
|-------|-------|--|------|
| IMPL | 63:60 | Implementation number of this host adapter | R |
| REV | 59:56 | Revision number for this design | R |
| DE | 01 | Diagnostic mode enable. Set to 1 to enable diagnostic mode access. This bit is reset to 0 | R/W |
| SB_EN | 00 | Streaming cache enable/disable. Set to 1 to enable the streaming cache. This bit is reset to 0 | R/W |

4.6.2 Streaming Cache Page Invalidate/Flush Register

This is a write-only pseudo register. It provides a means for software to cause an entry in the streaming cache with a matching tag to become invalidated/flushed. The data written to this address contains the virtual page number to be used for match comparison. The flush/invalidation is based on an 8K page size.

Table 4-33 Streaming Cache Page Invalidate/Flush Register

| Field | Bits | Description | Type |
|----------|-------|---|------|
| FLUSH_A | 31:13 | 8K virtual page to be invalidated/flushed | W |
| Reserved | 12:0 | These bits are ignored | W |

4.6.3 Streaming Cache Flush Synchronization Register

The Flush Synchronization Register enables software to determine when flush data has entered the coherent memory domain. Data written to this register contains the physical address of the flush flag. Writing to this register triggers the U2S to set the flush flag to (0x1), when it has completed all in-progress flush operations. The low order two bits of the FLAG_PA address is ignored. See Chapter 14, “Streaming Cache,” for a further description of the synchronization.

Note: Properly functioning hardware *MUST* set the flush flag within 0.5 seconds. Software may use this value as a time-out to protect against catastrophic failure.

Table 4-34 Streaming Cache Flush Synchronization Register

| Field | Bits | Description | Type |
|----------|-------|--|------|
| FLAG_PA | 40:02 | Word aligned physical address for synchronization update | W |
| Reserved | 01:00 | This bit is ignored | W |

The FLAG_PA must point to a valid CACHEABLE address. No hardware checking is done to ensure this and hardware will generate a cacheable transaction regardless of the address. If the address is non-cacheable, the error would be logged in the USC using the SYSIO_Status Register (IAddr bit).

4.6.4 Streaming Cache Page Tag Diagnostic Access

The Page Tags are directly accessible through PIO access. This can be done only when the DE bit of the Stream Cache Control Register is set to one.

Table 4-35 Streaming Cache Page Tag Format

| Field | Bits | Description | Type |
|-------|-------|--------------------------------------|------|
| PTPA | 48:21 | Physical page number (as an 8K page) | R/W |
| PTVA | 20:02 | Virtual page number (as an 8K page) | R/W |
| PTVD | 01 | Valid bit for page | R/W |
| PTRD | 00 | Read/Write bit for page | R/W |

Caution: Valid bits on all entries should be reset to 0 after finishing diagnostics of the page tag.

4.6.5 Streaming Cache Line Tag Diagnostic Access

The line tag contains information related to the line in the streaming cache. This information can be directly accessed when the streaming cache is in the diagnostic mode, and the DE bit is set in the Stream Cache Control Register.

Table 4-36 Streaming Cache Line Tag Format

| Field | Bits | Description | Type |
|-------|-------|---|------|
| LTSP | 20:15 | Start pointer for dirty data portion of the cache | R/W |
| LTLA | 14:08 | Line offset address for this entry within an 8K page. Corresponds to VA[12:6] from the SBus access. The VPN is stored in the page tag array | R/W |
| LTEP | 07:02 | End pointer (+1) for dirty data portion | R/W |
| LTVD | 01 | Valid bit for line | R/W |
| LTFH | 00 | Fetch outstanding/flush necessary bit | R/W |

The LTEP field should be set to 0 if the page is readable. If writable, this field should be set to one greater than the end byte address of the dirty data chunk in the data RAM (modulo buffer size of 64 bytes).

Caution: Valid bits on all entries should be reset to 0 after finishing diagnostics of the line tag.

4.6.6 Streaming Cache Data RAM Diagnostic Access

Note: For data and error status accesses, bits 9:6 of the address select the cache number and bits 5:3 select the 8-byte entry (or 1-bit error status) within the cache.

There are sixteen 64-byte entries in the streaming cache. Physical address bits 9:6 select the entry number and bits 5:3 select the 8-byte quantity to access in the entry.

Table 4-37 Streaming Cache Data RAM Content Format

| Field | Bits | Description | Type |
|-------|-------|-------------|------|
| DRDA | 63:00 | Data | R/W |

4.6.7 Streaming Cache Error Status Diagnostic Access

Each entry of the streaming cache has eight error bits associated with it. Each error bit represents the error status of eight bytes of data. These bits are only visible to software during the diagnostic mode. Each error bit is accessed individually.

Table 4-38 Streaming Cache Data RAM Error Format

| Field | Bits | Description | Type |
|-------|------|--------------------------|------|
| DRER | 00 | UPA read reply error bit | R/W |

4.7 Interrupt State Registers

All interrupt registers are accessed as 64-bit words. Writing to an unused data field has no effect. Unspecified data bits are read back with 0. Table 4-39, Table 4-41, and Table 4-45 respectively list the addresses of the interrupt state registers: one for interrupt mapping registers, one for clear interrupt registers, and one for diagnostic interrupt registers.

4.7.1 Interrupt Mapping Registers

Table 4-39 Physical Address of Interrupt Mapping Registers

| Register | Physical Address | Access Size |
|--|------------------|-------------|
| SBus slot 0 interrupt mapping register | 0x1FE.0000.2C00 | 8 bytes |
| SBus slot 1 interrupt mapping register | 0x1FE.0000.2C08 | 8 bytes |
| SBus slot 2 interrupt mapping register | 0x1FE.0000.2C10 | 8 bytes |
| SBus slot 3 interrupt mapping register | 0x1FE.0000.2C18 | 8 bytes |
| SCSI interrupt mapping register | 0x1FE.0000.3000 | 8 bytes |
| Ethernet interrupt mapping register | 0x1FE.0000.3008 | 8 bytes |
| Parallel port interrupt mapping register | 0x1FE.0000.3010 | 8 bytes |
| Audio interrupt mapping register | 0x1FE.0000.3018 | 8 bytes |
| Power fail interrupt mapping register | 0x1FE.0000.3020 | 8 bytes |
| Keyboard/mouse/serial interrupt mapping register | 0x1FE.0000.3028 | 8 bytes |
| Floppy interrupt mapping register | 0x1FE.0000.3030 | 8 bytes |
| Thermal warning interrupt mapping register | 0x1FE.0000.3038 | 8 bytes |
| Keyboard interrupt mapping register ^[1] | 0x1FE.0000.3040 | 8 bytes |
| Mouse interrupt mapping register | 0x1FE.0000.3048 | 8 bytes |
| Serial interrupt mapping register | 0x1FE.0000.3050 | 8 bytes |
| Timer 0 interrupt mapping register | 0x1FE.0000.3060 | 8 bytes |
| Timer 1 interrupt mapping register | 0x1FE.0000.3068 | 8 bytes |
| UE interrupt mapping register | 0x1FE.0000.3070 | 8 bytes |
| CE interrupt mapping register | 0x1FE.0000.3078 | 8 bytes |

4. Programming Model

Table 4-39 Physical Address of Interrupt Mapping Registers

| Register | Physical Address | Access Size |
|---|--|-------------|
| SBus error interrupt mapping register | 0x1FE.0000.3080 | 8 bytes |
| Power management walk-up interrupt mapping register | 0x1FE.0000.3088 | 8 bytes |
| UPA expansion (graphics) interrupt mapping register | 0x1FE.0000.3090 and 0x1FE.0000.6000 ^[2] | 8 bytes |
| Reserved interrupt mapping register | 0x1FE.0000.3098 and 0x1FE.0000.8000 ^[2] | 8 bytes |

1. The keyboard, mouse, and serial interrupts are defined for future devices which do not combine all of the interrupts into one.
2. Accesses to either of these addresses behave identically; in other words, the registers are double mapped.

Interrupts delivered to the processor by the U2S have the format shown in the following illustration:

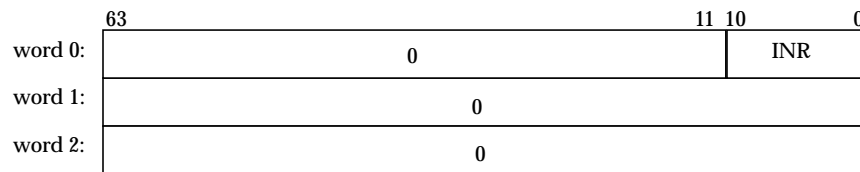


Figure 4-2 Interrupt Format Delivered to the Processor

INR is an 11-bit interrupt number which indicates the cause of the interrupt. Where possible, the interrupt is precise (i.e., it points to only one interrupt source). This permits the dispatch of the proper interrupt service routine without any register polling. Bits 11 through 63 of the first word are guaranteed to be 0 for all U2S-generated interrupts. Software can use this knowledge to distinguish these interrupts from others such as cross-calls. Words 1 and 2 are guaranteed to be 0.

To conform to the format shown above, interrupt mapping registers are supplied to specify those bits of the INR which are not hard coded. The two formats of mapping registers are shown below. For the partial format, the lower six bits are not writable and return the appropriate INO when read. In the case of the SBus interrupts, the INO returned is 0. The value of IGN comes from the U2S Control Register. See Table 4-2, “U2S Control Register Definition,” on page 26 for details.

Note: All U2S internal interrupts and SBus interrupts use the partial format. The full format is only used by the graphics interrupts.

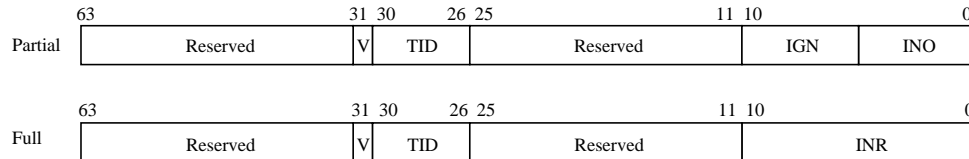


Figure 4-3 Interrupt Format of Word 0

Graphics interrupts have a different format. They are full because the U2S is acting only as the interrupt concentrator for the graphics interrupt. Logically, the interrupt and its associated state reside in the graphics device. Only the mapping register exists in the U2S. Having full access to the INR enables software to provide consistency in programming the upper bits for various devices. For example, software might put the PortID of a device in these upper bits. Since the graphics device is a different port than the U2S, the value can be programmed independently.

V (RW)

Valid bit: When set to 0, valid bit inhibits the dispatch of the interrupt to the processor. It does not affect state. In other words, an interrupt in the transmit state will stay there as long as valid is 0 without sending the interrupt. As soon as valid is set to 1, the interrupt will be delivered. Interrupts in the pending state can still be forced back to idle by writing the interrupt-pending register regardless of the state of valid. The power-up state of this bit is 0.

TID (RW)

Target ID: Target ID is the UPA Port ID (PID) of the target which is to receive the interrupt.

IGN (R)

Interrupt Group Number: The IGN bit is not writable through the mapping registers, and instead, is set in the U2S Control Register.

INO (R)

Interrupt Number Offset: For the on-board input/output (OBIO) devices and error interrupts, the INO field is as specified in Table 4-40, "INO Assignments," on page 59.

Table 4-40 INO Assignments

| INO | Interrupt Source |
|--------|---------------------------------------|
| 000nnn | SBus slot 0 (nnn = level 1 through 7) |
| 001nnn | SBus slot 1 (nnn = level 1 through 7) |
| 010nnn | SBus slot 2 (nnn = level 1 through 7) |
| 011nnn | SBus slot 3 (nnn = level 1 through 7) |
| 100000 | SCSI |
| 100001 | Ethernet |
| 100010 | Parallel port |
| 100100 | Audio (APC) |
| 100101 | Power failure |
| 101000 | Keyboard and serial ports |
| 101001 | Floppy |
| 101010 | Thermal warning |
| 101011 | Keyboard (reserved) ^[1] |
| 101100 | Mouse (reserved) ^[1] |
| 101101 | Serial (reserved) ^[1] |
| 110000 | Timer/Counter 0 |
| 110001 | Timer/Counter 1 |
| 110100 | UE |
| 110101 | CE |
| 110110 | SBus asynchronous error |
| 110111 | Power management wakeup |
| 111111 | Reserved |

1. These interrupt numbers are reserved for a future implementation in which the keyboard, mouse, and serial interrupts are not wired together.

Note: For the SBus interrupts, there is only one mapping register per bus. However, because the interrupt level is part of the value delivered in the interrupt vector, SBus interrupts are still precise.

INR (RW)

Interrupt Number: The full field is programmable for the on-board graphics and UPA slave slot interrupts.

4.7.2 Clear Interrupt Registers

The clear Interrupt Pseudo Registers are shown in Table 4-41.

Table 4-41 Physical Address of Clear Interrupt Pseudo Registers

| Register | Physical Address | Access Size |
|--|----------------------|-------------|
| SBus slot 0 clear interrupt register | 0x1FE.0000.3408 – 38 | 8 bytes |
| SBus slot 1 clear interrupt register | 0x1FE.0000.3448 – 78 | 8 bytes |
| SBus slot 2 clear interrupt register | 0x1FE.0000.3488 – b8 | 8 bytes |
| SBus slot 3 clear interrupt register | 0x1FE.0000.34c8 – f8 | 8 bytes |
| SCSI clear interrupt register | 0x1FE.0000.3800 | 8 bytes |
| Ethernet clear interrupt register | 0x1FE.0000.3808 | 8 bytes |
| Parallel port clear interrupt register | 0x1FE.0000.3810 | 8 bytes |
| Audio clear interrupt register | 0x1FE.0000.3818 | 8 bytes |
| Power fail clear interrupt register | 0x1FE.0000.3820 | 8 bytes |
| Keyboard/mouse/serial clear interrupt register | 0x1FE.0000.3828 | 8 bytes |
| Floppy clear interrupt register | 0x1FE.0000.3830 | 8 bytes |
| Thermal warning clear interrupt register | 0x1FE.0000.3838 | 8 bytes |
| Keyboard clear interrupt register | 0x1FE.0000.3840 | 8 bytes |
| Mouse clear interrupt register | 0x1FE.0000.3848 | 8 bytes |
| Serial clear interrupt register | 0x1FE.0000.3850 | 8 bytes |
| Timer 0 clear interrupt register | 0x1FE.0000.3860 | 8 bytes |
| Timer 1 clear interrupt register | 0x1FE.0000.3868 | 8 bytes |
| UE clear interrupt register | 0x1FE.0000.3870 | 8 bytes |
| CE clear interrupt register | 0x1FE.0000.3878 | 8 bytes |
| SBus asynchronous error clear interrupt register | 0x1FE.0000.3880 | 8 bytes |
| Power management wakeup clear interrupt register | 0x1FE.0000.3888 | 8 bytes |

One register exists per interrupt source. The lower two bits of the data word written to this register specify the type of operation as shown in Table 4-42, "Clear Interrupt Pseudo Register." All other bits should be written as 0 to guarantee future compatibility. Reads of these registers return 0.

Table 4-42 Clear Interrupt Pseudo Register

| Data Value | Description |
|------------|--|
| 0x0 | Transition the state machine from any state to idle |
| 0x1 | Transition from any state to transmit |
| 0x2 | Reserved |
| 0x3 | Transition the state machine from any state to pending |

Note: Interrupts can be forced by writing, 0x1, to the clear interrupt pseudo register. To determine the interrupt state, use the interrupt state diagnostic registers.

4.7.3 Interrupt Retry Timer Register

Table 4-43 Physical Address of Interrupt Retry Timer Registers

| Register | Physical Address | Access Size |
|--------------------------|------------------|-------------|
| Interrupt retry register | 0x1FE.0000.2C20 | 8 bytes |

After an interrupt packet receives a NACK from the USC, the U2S will wait for a certain number of clocks and reissue again. This register controls the number of clocks the interrupt dispatch unit should wait before reissuing the interrupt packet. The count specified by this register is not precise: it is a free-running counter which the logic samples. It must roll through 0 twice before the packet is retried.

Table 4-44 Interrupt Retry Timer Register

| Field | Bits | Description | Type |
|-------|------|----------------------------|------|
| LIMIT | 7:0 | Limit - the retry interval | R/W |

Note: The retry timer provides a maximum of $(255 + 256)$ clocks of delay before reissuing the interrupt to the UPA. The maximum delay is approximately 10 μsec using the internal U2S clock for a reference source (5.1 μsec per iteration through the counter with a worst case of nearly two complete cycles counting to 0). The minimum delay is $(\text{LIMIT} + 1)$ clock cycles.

4.7.4 Interrupt State Diagnostic Registers

Table 4-45 Physical Address of Interrupt State Diagnostic Registers

| Register | Physical Address | Access Size | Type |
|--|------------------|-------------|------|
| SBus interrupt state diagnostic register | 0x1FE.0000.4800 | 8 bytes | R |
| OBIO and miscellaneous interrupt state diagnostic register | 0x1FE.0000.4808 | 8 bytes | R |

Each interrupt input has a state register associated with it. This state register can be either of type “level” or “pulse.”

In the level-sensitive case, three states are present: idle, transmit, and pending. Idle represents the state where no interrupts are reported. Transmit indicates that an interrupt has been detected and should be delivered to the processor if the valid bit is set for the mapping register. Pending is the state when the interrupt has been delivered to the processor and subsequent interrupt conditions are filtered (ignored) until software pushes the state machine back to idle.

In the pulse case, two states are present: idle and transmit. Transmit has the same meaning as for the level-sensitive case. No pending state is present so the state machine transitions from transmit back to idle.

The interrupt dispatch unit uses information in these registers to control the dispatching of interrupts. Diagnostic access is provided to allow software to read the state of each interrupt source. The clear interrupt registers provide individual write paths to the interrupt sources. Please read the Interrupt section for a more detailed description about interrupt dispatching and state transition. The meaning of the state bits and their layout are shown in Table 4-46, “Interrupt State Meaning.”. For the case of the SBus Interrupt Diagnostic registers, the formula for computing the location of the interrupt state is given by:

$\text{Int_State}(\text{slot } m, \text{level } n) \text{ is at location } (16m + 2n + 1):(16m + 2n)$

Table 4-46 Interrupt State Meaning

| Field | Description |
|----------------|--|
| INT_STATE[1:0] | 00 - Idle state; no interrupt received or pending 01 - Transmit state; interrupt is received but not dispatched 11 - Pending state; interrupt is received and dispatched 10 - Illegal state |

Definitions of the registers are shown in a general way in Table 4-47. See the formula above for specific bit positions. As an example, the bit position for SBus slot 1, level 2 is [21:20].

Table 4-47 SBus Internal Diagnostic Register Definition

| Bits | Description |
|-------|-----------------------|
| 1:0 | Reserved |
| 15:2 | SBus slot 0 level 1-7 |
| 17:16 | Reserved |
| 31:18 | SBus slot 1 level 1-7 |
| 33:32 | Reserved |
| 47:34 | SBus slot 2 level 1-7 |
| 49:48 | Reserved |
| 63:50 | SBus slot 3 level 1-7 |

Table 4-48 OBIO and Miscellaneous Internal Diagnostic Register Definition

| Bits | Description |
|------|------------------------------|
| 1:0 | SCSI internal state |
| 3:2 | Ethernet internal state |
| 5:4 | Parallel port internal state |
| 7:6 | Audio internal state |
| 9:8 | Power fail internal state |

4. Programming Model

Table 4-48 OBIO and Miscellaneous Internal Diagnostic Register Definition

| Bits | Description |
|-------|--|
| 11:10 | Keyboard/mouse/serial internal state |
| 13:12 | Floppy internal state |
| 15:14 | Thermal warning internal state |
| 17:16 | Keyboard internal state |
| 19:18 | Mouse internal state |
| 21:20 | Serial internal state |
| 23:22 | Timer 0 Internal state |
| 25:24 | Timer 1 internal state |
| 27:26 | UE internal state |
| 29:28 | CE internal state |
| 31:30 | SBus error internal state |
| 33:32 | Power management wakeup internal state |
| 34 | Reserved internal state |
| 35 | Expansion UPA internal state |
| 63:36 | Reserved (return 0 on read) |

4.8 Counter/Timer Registers

Table 4-49 Physical Address of Counter/Timer Registers

| Register | Physical Address | Access Size |
|--------------------------------|------------------|-------------|
| Timer/Counter 0 count register | 0x1FE.0000.3C00 | 8 bytes |
| Timer/Counter 0 limit register | 0x1FE.0000.3C08 | 8 bytes |
| Timer/Counter 1 count register | 0x1FE.0000.3C10 | 8 bytes |
| Timer/Counter 1 limit register | 0x1FE.0000.3C18 | 8 bytes |

Note: The WAKEUP_EN bit in the SBus Control register results in an increment once every 1000 clocks rather than once per clock.

4.8.1 Count Registers

Table 4-50 Count Register

| Field | Bits | Description | Type |
|----------|-------|---|------|
| Reserved | 63:29 | Reserved, read as 0 | R |
| Count | 28:0 | Value to preset counter on write, and current count value on read | R/W |

Each Count Register provides the means to load the timer with a preset value on write, and return the current value of the timer on read. Count Register increments once per microsecond.

4.8.2 Limit Registers

Table 4-51 Limit Register

| Field | Bits | Description | Type |
|----------|-------|--|------|
| Reserved | 63:32 | Reserved, read as 0 | R |
| INT_EN | 31 | Enable interrupt from this timer. Reset to 0 at power-up | R/W |
| Reload | 30 | Writes to the LIMIT register with this bit set causes the counter to restart at 0x0. Read as 0 | W |
| Periodic | 29 | When set, causes the counter to reset to 0x0 when LIMIT is reached | R/W |
| Limit | 28:0 | Counter interrupt comparison value | R/W |

Each Limit Register provides the means to enable and disable the interrupt, re-loading the counter, setting periodic interrupt, and setting LIMIT for a counter time-out comparison.

4.9 Performance Monitor Registers

Table 4-52 Physical Address of Performance Monitor Registers

| Register | Physical Address | Access Size |
|--------------------------------------|------------------|-------------|
| Performance monitor control register | 0x1FE.0000.0100 | 8 bytes |
| Performance counter register | 0x1FE.0000.0108 | 8 bytes |

In order to gather useful statistics on the performance of the U2S, a pair of registers provide counts of key events. There are only two counters present, and the control register selects the input for each of the counters.

4.9.1 Performance Monitor Control Register

This register controls the events to be monitored by the Performance Counter Register. The event counters in the Performance Counter Register will be reset when the respective CLR[0,1] bits are written with a 1.

Table 4-53 Performance Monitor Control Register

| Field | Bits | Description | Type |
|----------|-------|--|------|
| Reserved | 63:16 | Reserved, read as 0 | R |
| CLR1 | 15 | Clears the counter indicated by SEL1 | W |
| Reserved | 14:12 | Reserved, read as 0 | R |
| SEL1 | 11:8 | Select event source for counter 1. Selected source counter is cleared when the CLR1 field is written | R/W |
| CLR0 | 7 | Clears the counter indicated by SEL0 | W |
| Reserved | 6:4 | Reserved, read as 0 | R |
| SEL0 | 3:0 | Select event source for counter 0. Selected source counter is cleared when the CLR0 field is written | R/W |

Table 4-54 defines the code to select monitored events in the U2S.

Table 4-54 Performance Counter Event Sources

| SEL0, SEL1 | Event Sources |
|------------|--|
| 0x0 | Number of streaming DVMA read transfers |
| 0x1 | Number of streaming DVMA write transfers |
| 0x2 | Number of consistent DVMA read transfers |
| 0x3 | Number of consistent DVMA write transfers |
| 0x4 | Number of TLB misses |
| 0x5 | Number of streaming buffer read misses |
| 0x6 | Number of SBus cycles. SBus is granted to DVMA |
| 0x7 | Number of bytes transferred using DVMA |
| 0x8 | Number of interrupts |
| 0x9 | Number of interrupt NACK on UPA |
| 0xA | Number of PIO read transfers |
| 0xB | Number of PIO write transfers |
| 0xC | Number of SBus reruns |
| 0xD | Number of SBus cycles. SBus is consumed by PIO |
| 0xE-0xF | Reserved. Counter value is undefined when these sources are chosen |

4.9.2 Performance Counter Register

This is a 64-bit read-only register. Value read back contains the counts of two events selected by the Performance Monitor Control Register. The two counters operate independently. Values for both counters are sampled on the same cycle. When the counter reaches its maximum count, it will wrap around to 0x0 and continue counting. Software needs to detect and handle the overflow condition.

Table 4-55 Performance Counter Register

| Field | Bits | Description | Type |
|------------|-------|------------------------------------|------|
| CNT0[31:0] | 63:32 | Contains value for event counter 0 | R |
| CNT1[31:0] | 31:00 | Contains value for event counter 1 | R |

This chapter describes the PIO Decoder which decodes PIO Requests from the UPA and notifies the bus controller of the destination.

5.1 Definition of Terms

Prefixes to Signal Names

- U2S - UPA to SBus buses (address and data)
- S2U - SBus to UPA buses (address and data)
- DCC - Merge Buffer (was DMA Cache)
- MDU - Mondo Dispatch Unit
- MMU - IOMMU
- SBM - SBus Module
- STC - Streaming Cache

Example of Signal Names

- U2S_PA - UPA to SBus PIO Address bus
- U2S_PD - UPA to SBus PIO Data bus
- U2S_DA - UPA to SBus DMA Address bus
- U2S_DD - UPA to SBus DMA Data bus

Signals which are active low end in “_”, such as T0_RDY_.

5.2 Functional Description

The PIO Decoder decodes UPA packets. The address is decoded to determine the target for the PIO Read or Write. This information is sent to the Bus Controller. The size is decoded through the transaction type and the byte enables.

PIO to internal U2S blocks will always be 8 bytes. PIO to SBus can be 1, 2, 4, 8, 16, or 64 bytes.

5.2.1 PIO Transaction Flow

This section describes the flow of an 8 byte PIO request and reply through the U2S. See Figure 5-1 for each step (each clock/action is labeled by a number with in a circle). An 8 byte PIO request involves 1 cycle on the data busses. For multiple cycle PIO requests to the SBus, See Chapter 7, "Bus Controller," for more details.

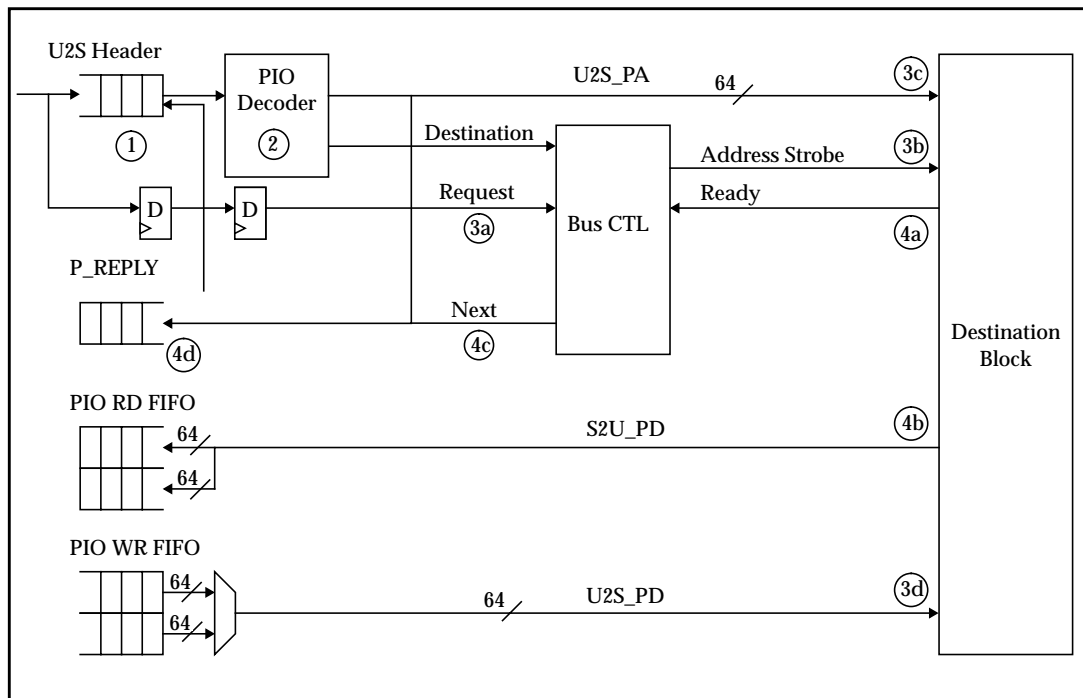


Figure 5-1 PIO Transaction Flow

PIO Transaction Flow (each clock cycle is identified by a number):

Clock cycle 1 - A PIO Read or Write request enters the U2S Header FIFO.

Clock cycle 2 - The UPA Packet is decoded; the size is determined; the destination is determined. In parallel, the request signal is passing through the asynchronous boundary.

Clock cycle 3a - The Bus Controller sees the request, the type, size, and destination of the transaction.

Clock cycle 3b - The Bus Controller asserts address strobe to the destination block.

Clock cycle 3c - The Bus Controller freezes the U2S_PA bus with the address, type, and size of the transaction.

Clock cycle 3d - For PIO Writes, the Bus Controller also freezes the first cycle of data on the U2S_PD bus.

Clock cycle 4a - When the block has completed the PIO request, it notifies the Bus Controller via a ready signal. For PIO Writes, the ready signal indicates that the block has taken the data.

Clock cycle 4b - For PIO Reads, the ready signal indicates that the block is driving the S2U_PD bus with the PIO Read data. The Bus Controller clocks the data into the UPA PIO Read Fifo.

Clock cycle 4c - The Bus Controller notifies the UPA that the PIO transaction is complete. This allows the UPA to advance the fifo and begin processing the next request.

Clock cycle 4d - The Bus Controller routes the request to the P_REPLY block.

Note: Only one PIO transaction is processed at a time in the U2S. For PIO Writes the data could have an ECC Uncorrectable Error. In this case, the Bus Controller simply clocks the data out of the data FIFO without transferring the data to the target block.

5.2.2 Byte Enables

All valid byte enable combinations are decoded. All PIO accesses must be aligned. That is, 2 byte accesses must have byte enables aligned on two-byte boundaries, 4 byte accesses must have byte enables aligned on four-byte boundaries, etc. Only 1,2,4,8,16, and 64 byte requests are allowed. Invalid byte enables cause `usr_fe_` to be asserted, which in turn causes `P_RERR` to be sent as the `P_REPLY`. The byte enables also are decoded to determine the size of the PIO access, and are used to calculate `u2s_pa[3:0]` and encode `usr_sz`.

5.2.3 Size Encoding

The following table specifies the encoding of `usr_sz`.

Table 5-1 Size Encoding

| Size (Bytes) | <code>usr_sz[2:0]</code> |
|--------------|--------------------------|
| 1 | 000 |
| 2 | 001 |
| 4 | 010 |
| 8 | 011 |
| 16 | 100 |
| 64 | 101 |

5.2.4 Valid Transactions

The following table lists all the transactions that are decoded to be valid.

Table 5-2 Valid Slave Transactions

| Transaction | Description |
|--------------------------|---|
| <code>P_NCRD_REQ</code> | Non-cached Read Request (1,2,4,8,16 Bytes) |
| <code>P_NCBRD_REQ</code> | Non-cached Block Read Request (64 Bytes) |
| <code>P_NCWR_REQ</code> | Non-cached Write Request (1,2,4,8,16 Bytes) |
| <code>P_NCBWR_REQ</code> | Non-cached Block Write Request (64 Bytes) |

All other transactions are considered invalid and `P_RERR` is returned as the `P_REPLY`.

5.2.5 Other Errors

A parity error from the UPA header will cause P_FERR is returned as the P_REPLY.

Only 8 byte PIO accesses are allowed to internal registers. Non 8 byte PIO accesses to internal registers will be treated as an invalid transaction. And an access to an invalid address will also be treated as an invalid transaction.

This chapter describes the DMA Controller which schedules and keeps track of all outstanding DMA requests.

6.1 Definition of Terms

Prefixes to Signal Names

- U2S - UPA to SBus buses (address and data)
- S2U - SBus to UPA buses (address and data)
- DCC - Merge Buffer (was DMA Cache)
- MDU - Mondo Dispatch Unit
- MMU - IOMMU
- SBM - SBus Module
- STC - Streaming Cache

Example of Signal Names

- U2S_PA - UPA to SBus PIO Address bus
- U2S_PD - UPA to SBus PIO Data bus
- U2S_DA - UPA to SBus DMA Address bus
- U2S_DD - UPA to SBus DMA Data bus

Signals which are active low end in “_”, such as T0_RDY_.

6.2 Overview

The DMA Controller is responsible for keeping track of all DMA read and write requests that come in from any requesting source in the U2S destined for the UPA. These sources include the SBus Module, the Streaming Cache, the Mondo Dispatch Unit (Interrupts are considered as DMA Writes), and the IOMMU.

6.2.1 DMA Controller Overview

The DMA Controller is composed of three subblocks: the DMA Request Encoder, the DMA Reply Controller, and the DMA Scoreboard.

The DMA Request Encoder builds the UPA header packet for all sources that make DMA (or Interrupt) requests and inserts the transaction into the DMA Scoreboard.

The DMA Reply Controller has two main functions:

1. Retiring the DMA request from the scoreboard; and,
2. Participating in the internal arbitration sequence to deliver the data back to the requesting block for DMA read replies.

The DMA Scoreboard's main function is to keep a record of all outstanding DMA transactions to the UPA. Note that table walks from the IOMMU are considered to be a DMA Read.

The other block that is heavily involved with DMA traffic is the merge buffer. Briefly, the merge buffer is necessary for cache coherent writes to main memory that are of less than 64 bytes. Since transfers to memory are defined only in 64 bytes, a smaller transfer (1, 2, 4, 8, 16, or 32 bytes) from SBus must be merged with existing data and written out as a 64 byte block. This requires a 64 byte block to be read from memory and merged with the new data, (see Chapter 11, "DMA Merge Buffer," for a full discussion).

6.2.2 DMA Transaction Flow

This section describes the flow of a DMA request and reply through the DMA Controller. Refer to figures for each step (each clock/action is labeled by a number within a circle). Note that the U2S only uses one class (class 0) for all its transactions; the other class is not used.

The following discussion uses a simplified signal handshake protocol between the requesting DMA source and the Bus Controller, (for more details see Chapter 7, “Bus Controller”).

The following discussion concerns only the address flow.

6.2.2.1 DMA Write to IO Space

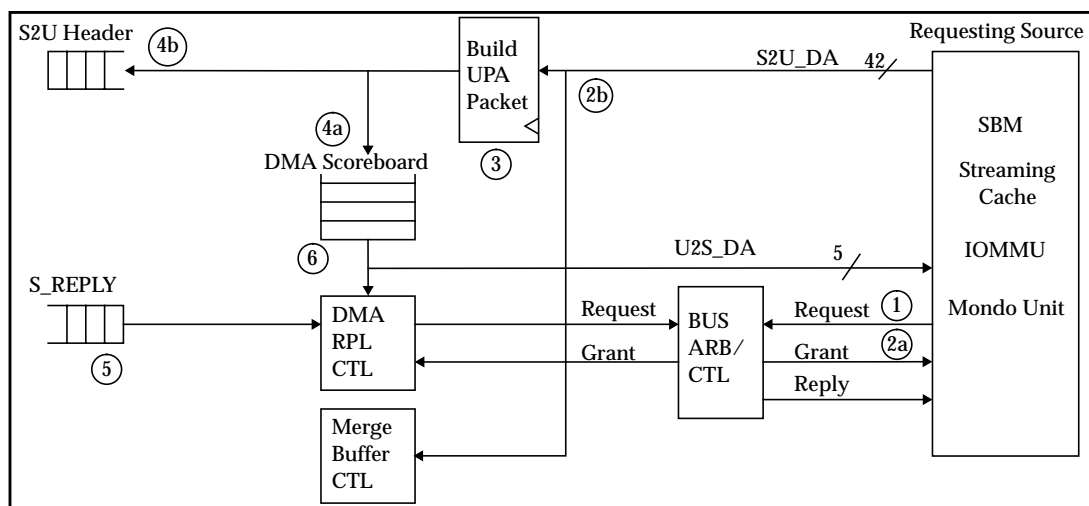


Figure 6-1 DMA Write to IO Space (Address only)

DMA Write to IO space (each clock cycle is identified by a number):

Note: The SBM is the only block which will access IO space.

- Clock cycle 1 - The SBM raises a request for a transfer. It indicates a write with the C bit clear. The Bus Controller begins arbitration
- Clock cycle 2a - The SBM wins arbitration
- Clock cycle 2b - The SBM drives the address bus (and data busses)

- Clock cycle 3 - The UPA header is formed. When accessing IO Space, there are 2 cases: 1) if the write size is 1, 2, 4, 8, 16, or 64 bytes, then the write is directly translated into a single UPA write; 2) if the write size is 32 bytes, it is broken up into two individual 16 byte writes, since 32 byte transfers are not supported on UPA. Note that for reads, regardless of size, it is sent out as a 64 byte read, with the extra data ignored
- Clock cycle 4a - The Request is entered in the DMA scoreboard. If the write is broken down into two 16 Byte writes, then the second write will be entered into the DMA Scoreboard next cycle
- Clock cycle 4b - The Request is sent to the UPA interface. If the write is broken down into two 16 Byte writes, the second write will be sent to the UPA interface next cycle
- Clock cycle 5 - For writes, S_REPLY indicates that the data has been taken
- Clock cycle 6 - The DMA Reply Controller is informed of the reply. It extracts the transaction from the scoreboard

6.2.2.2 DMA Read Request

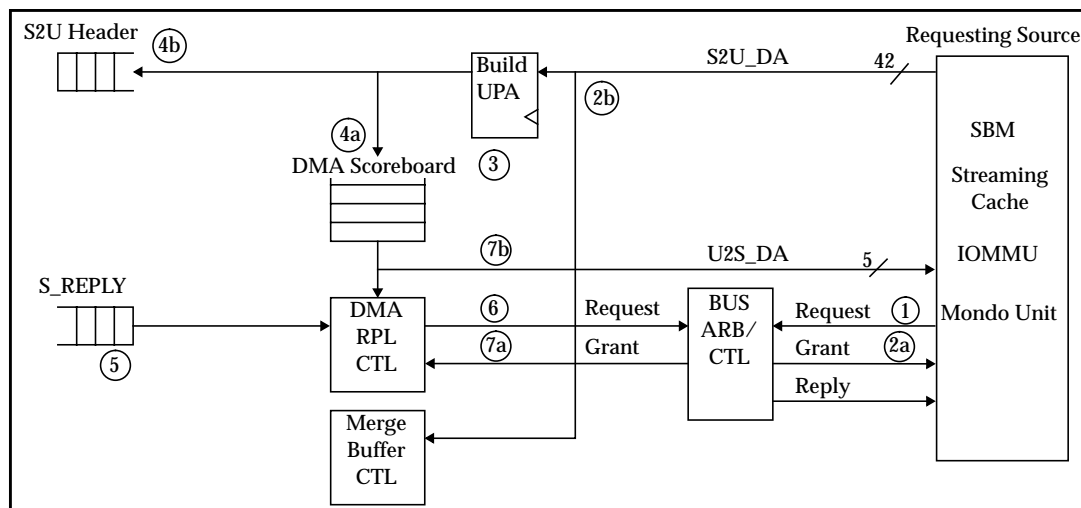


Figure 6-2 DMA Read Request to Memory (Address Only)

DMA Read Request to Memory or IO Space (each clock cycle is identified by a number):

Note: The SBM is the only block which will access IO space.

Clock cycle 1 - The block raises a request for a transfer. It indicates a read. The Bus Controller begins arbitration.

Clock cycle 2a - The block wins arbitration.

Clock cycle 2b - The block drives the address bus.

Clock cycle 3 - The UPA header is formed. For DMA reads, regardless of size, it is sent out as a 64 byte read, with the extra data ignored upon return.

Clock cycle 4a - The Request is entered in the DMA Scoreboard.

Clock cycle 4b - The Request is sent to the UPA interface.

Clock cycle 5 - For reads, S_REPLY indicates that the data has arrived.

Clock cycle 6 - The DMA Reply Controller is informed of the reply. It raises a request to return the data. The Bus Controller begins arbitration.

Clock cycle 7a - The DMA Controller wins arbitration.

Clock cycle 7b - The DMA Reply Controller extracts the transaction from the scoreboard. If the read was from the Streaming Cache, the DMA Controller sends a tag back.

The data can be transferred in this cycle as well. Extra data is clocked out of the UPA Read FIFOs by the Bus Controller.

6.2.2.3 64 Byte DMA Write Request to Memory

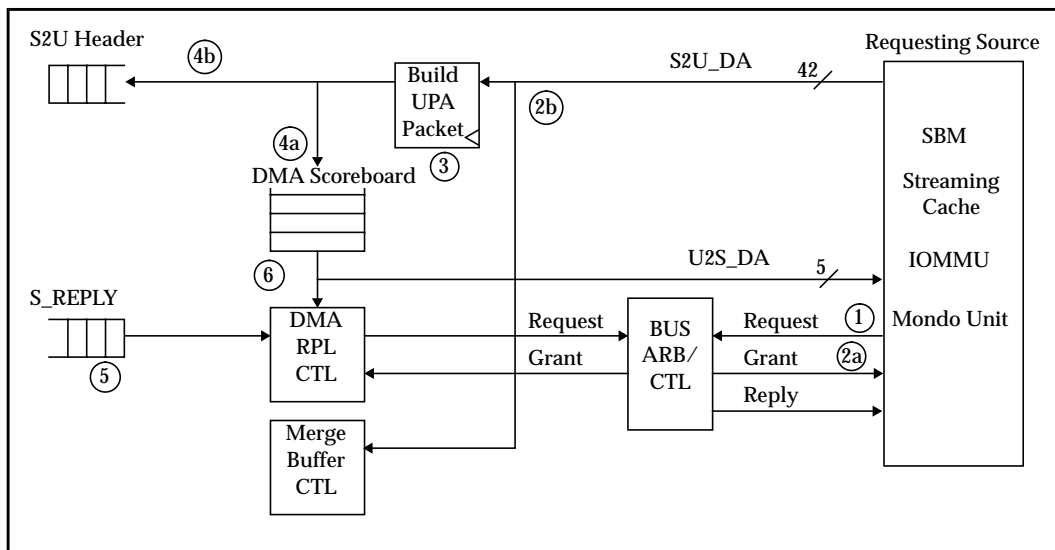


Figure 6-3 64 Byte DMA Write Request to Memory

64 Byte DMA Write Request to Memory or Interrupt Packet (each clock cycle is identified by a number):

Clock cycle 1 - The block raises a request for a transfer. It indicates a 64 byte write with the C bit set. The Bus Controller begins arbitration.

Clock cycle 2a - The block wins arbitration.

Clock cycle 2b - The block drives the address bus (and data busses).

Clock cycle 3 - The UPA header is formed. 64 Byte writes bypass the merge buffer, and are sent out to the UPA as a Write-Invalidate.

Clock cycle 4a - The Request is entered in the DMA Scoreboard.

Clock cycle 4b - The Request is sent to the UPA interface.

Clock cycle 5 - For writes, S_REPLY indicates that the data has been taken.

Clock cycle 6 - The DMA Reply Controller is informed of the reply. It extracts the transaction from the scoreboard.

6.2.2.4 Less than 64 Byte DMA Write to Memory

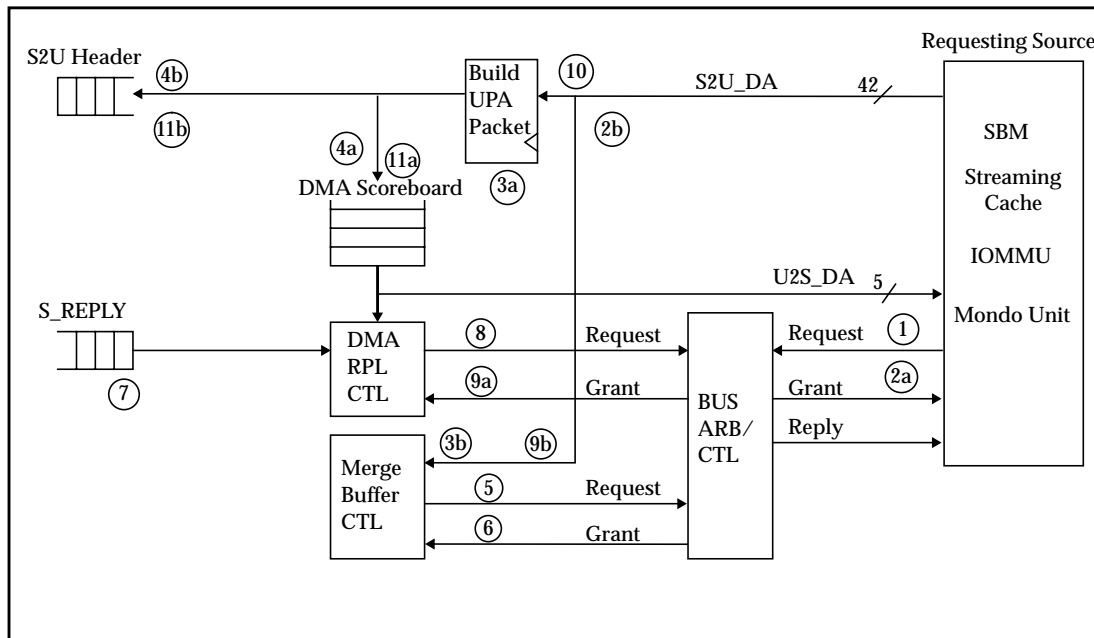


Figure 6-4 Less than 64 Byte DMA Write to Memory

Less than 64 Byte DMA Write Request to Memory (each clock cycle is identified by a number):

Clock cycle 1 - The block raises a request for a transfer. It indicates a less than 64 byte write with the C bit set. The Bus Controller begins arbitration.

Clock cycle 2a - The block wins arbitration.

Clock cycle 2b - The block drives the address bus (and data buses).

Clock cycle 3a - The UPA header is formed. The UPA Encoder block recognizes that it is a less than 64 byte write to memory, so it forms a Read-to-Own transaction for the Merge Buffer.

Clock cycle 3b - The write is also sent to the Merge Buffer.

Clock cycle 4a - The Read-to-Own request is entered in the DMA Scoreboard.

Clock cycle 4b - The request is sent to the UPA interface.

Clock cycle 5 - The Merge Buffer stalls all DMA activity to the UPA and requests the internal busses for a Writeback.

Clock cycle 6 - The Bus Controller grants the Merge Buffer ownership.

Clock cycle 7 - S_REPLY indicates that the data has arrived. The DMA Reply Controller is informed of the arrival of the data.

Clock cycle 8 - The DMA Reply Controller makes a request to the Bus Controller to transfer the data to the Merge Buffer. The Bus Controller begins arbitration.

Clock cycle 9a - The DMA Reply Controller wins arbitration.

Clock cycle 9b - The data is delivered to the Merge Buffer.

Clock cycle 10 - The data, while being merged, is sent directly to the UPA for a Writeback. The Merge Buffer issues a Writeback.

Clock cycle 11a - The Writeback is entered into the DMA Scoreboard.

Clock cycle 11b - The Writeback is sent to the UPA. After the Writeback completes, DMA Activity is allowed to proceed.

For a more complete description of the merge buffer operation see Chapter 11, "DMA Merge Buffer."

6.3 DMA Controller Functional Description

No arbitration occurs in the DMA Controller. All arbitration occurs in the Bus Arbitrator/Controller. The DMA Controller is then informed of the results, if a DMA request or reply is involved.

The two cases are:

1. A DMA Request wins arbitration: the DMA Request Encoder does this.
2. A DMA Reply wins arbitration: the DMA Reply Controller does this.

6.3.1 Overview

6.3.1.1 DMA Request Encoder

The DMA Request Encoder receives DMA Requests from internal blocks., delivered by the Bus Controller. It then translates the request into the appropriate UPA format. In the next cycle, it hands the packet to the S2U Header FIFO in the UPA block and to DMA Scoreboard.

The UPA Packet Encoder follows the following rules:

1. All reads to memory are issued as 64 Byte Read-to-Discards.
2. Reads to IO Space that are 16 bytes or less are issued as Read-to-Discard with the appropriate Byte Enables Set.
3. Reads to IO Space that are 32 or 64 bytes are issued as 64 Byte Read-to-Discards.
4. 64 Byte writes to memory are issued as a Write-Invalidate.
5. Less than 64 Byte writes to memory are issued as Read-to-Own on behalf of the Merge Buffer.
6. 32 Byte writes to IO Space are issued as two separate 16 Byte Writes.

6.3.1.2 DMA Reply Controller

The DMA Reply Controller, upon receiving the S_REPLY for a DMA transaction, retires the transaction from the DMA Scoreboard, and generates a request to the Bus Controller if data needs to be transferred.

6.3.1.3 DMA Scoreboard

The scoreboard is organized as a fifo since all transactions to the UPA are ordered. U2S only uses 1 master class, so all transactions to the UPA from U2S are ordered. There are two entries in the scoreboard.

| | | | | | | | | | |
|----|-----|-----|---------|------|-----|---|---|---|---|
| 53 | 52 | 51 | 48 | 47 | 7 | 6 | 4 | 3 | 0 |
| C. | RW. | Tag | Address | Size | Src | | | | |
| 1 | R | ? | 0x???? | 64 | STC | | | | |
| 1 | W | ? | 0x???? | 64 | SBM | | | | |

Figure 6-5 Examples of Scoreboard Entries

As shown in Figure 6-5 the scoreboard contains six fields:

1. C Bit (1 bit) - Cacheable bit from the IOMMU which indicates whether this transaction goes to memory or IO Space. Only the SBus (SBM) will make requests to IO Space.
2. RW (1 bit) - Read/Write bit.
3. Tag (4 bits) - Special field used by the STC only. This field will be returned to the source when the reply returns. It is a 4 bit field in which the requesting source can put any type of information. The STC will place an index number from its pool of buffers.
4. Address (41 bits) - in case of errors, the reply packet will contain the original address.
5. Size (3 bits) - size of transaction.
6. Source (4 bits) - the Bus Controller needs this for arbitration. This equals the destination encoding (see Chapter 7, "Bus Controller").

7.1 Definition of Terms

Prefixes to Signal Names

- U2S - UPA to SBus buses (address and data)
- S2U - SBus to UPA buses (address and data)
- DCC - Merge Buffer (was DMA Cache)
- MDU - Mondo Dispatch Unit
- MMU - IOMMU
- SBM - SBus Module
- STC - Streaming Cache
- TMR - Timer Counters

Example of Signal Names

- U2S_PA - UPA to SBus PIO Address bus
- U2S_PD - UPA to SBus PIO Data bus
- U2S_DA - UPA to SBus DMA Address bus
- U2S_DD - UPA to SBus DMA Data bus

7.2 Overview

The Bus Controller is responsible for arbitrating among all sources and destinations within the U2S. After choosing one or more winners, the Bus Controller steers all the multiplexers (MUXs) and enables all the latches, first in and first out (FIFO), and buffers necessary to transfer either the address or data or both from the source to the destination.

7.2.1 Bus Controller Overview

The Bus Controller is broken up into three subblocks: the U2S DMA Bus Controller, the S2U DMA Bus Controller, and the PIO Bus Controller. Each sub-block is responsible for arbitrating among different sources, choosing a winner, and finally steering the appropriate muxes to make the transfer. Sources which make requests include the Merge Buffer, Mondo Dispatch Unit, IOMMU, SBus Module, Streaming Cache, and the UPA. Requests will require the transfer of address and/or data to some destination. The destination of each request is checked to make sure that it has room and is ready to take the transfer. A source that makes a request to a destination that cannot take the transfer will not win arbitration.

The U2S DMA Bus Controller controls the U2S DMA Bus for transactions that come in from the UPA destined for any of the U2S blocks. These transactions include DMA Read Replies for the Merge Buffer (Read-to-Owns), IOMMU (Table Walks), SBus (DMA Reads), and Streaming Cache (DMA Read Prefetches).

The S2U DMA Bus Controller arbitrates and controls the S2U DMA Bus for transactions that need to go out to the UPA. Blocks that issue requests to the UPA include the Merge Buffer (Writebacks), IOMMU (Table Walks), Mondo Dispatch Unit (Interrupts), SBus (DMA Read and Writes), and Streaming Cache (DMA Read and Writes).

The PIO Bus Controller services PIO from the UPA.

7.2.2 Bus Architecture

There are four main buses in the U2S, two buses (PIO and DMA) in each direction:

- U2S_P (PIO Requests)
- S2U_P (PIO Replies)
- U2S_D (DMA Replies)
- S2U_D (DMA Requests)

These buses are shown in the following figures, see Figure 7-1 and Figure 7-2.

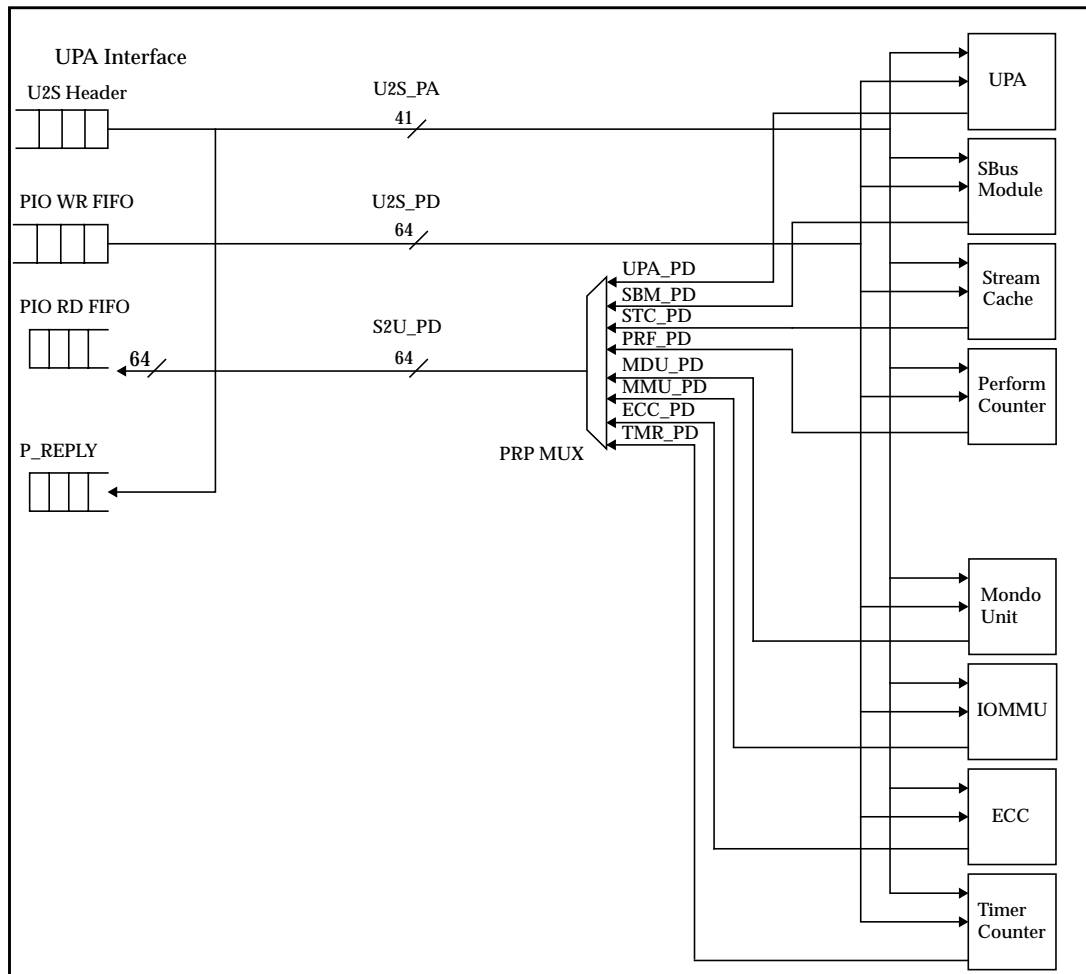


Figure 7-1 PIO Buses

The PIO Data buses are 64 bits wide. The U2S_PD bus is used for PIO Write data. The S2U_PD is used for PIO Read data. Both buses are frozen for the duration of a PIO transaction; so, only one PIO transaction may be serviced at a time.

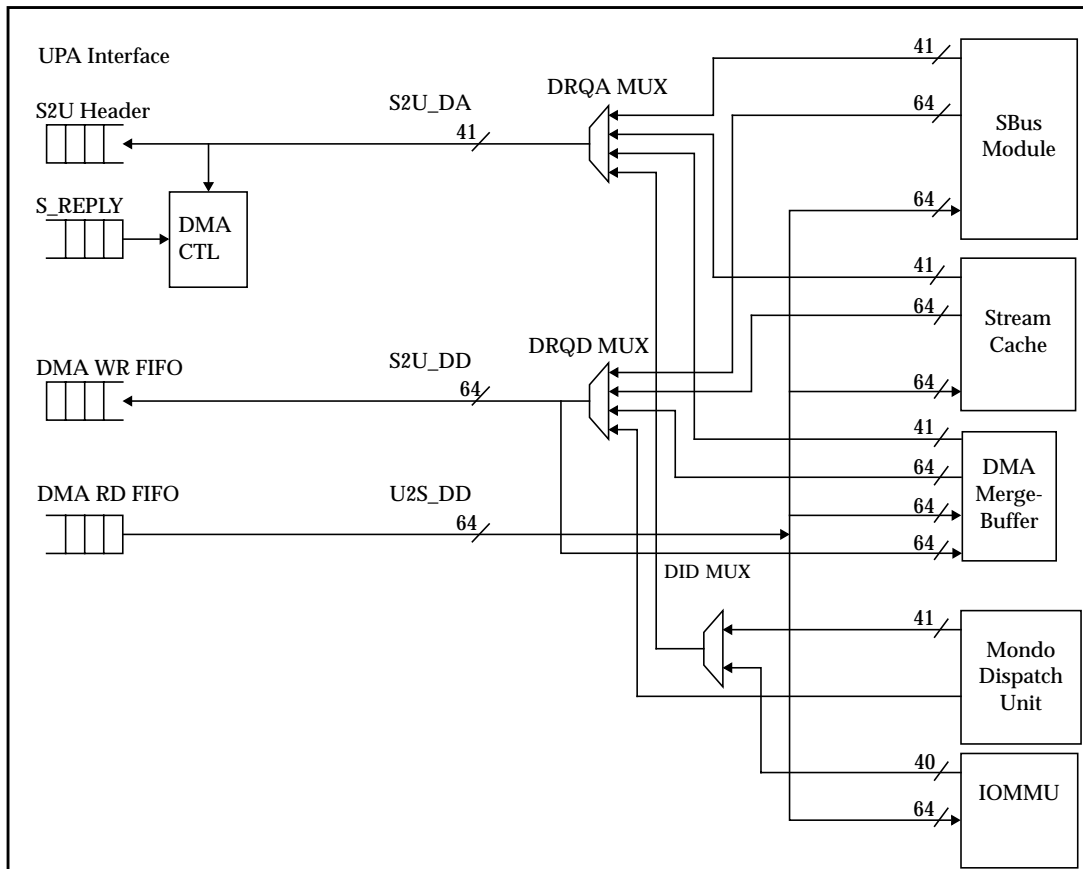


Figure 7-2 DMA Buses

The DMA Data buses are 64 bits wide. The S2U_DD bus is used for DMA writes, and the U2S_DD bus is used for DMA reads (replies).

7.3 Interface to Internal Blocks

This section describes the interface between the Bus Controller and the internal blocks. Section 7.3.1, “Generic Interface describes a generic interface which include several signals which are common across blocks. Section 7.3.2, “Specific Interfaces then describes how each specific internal block differs. Section 7.3.3, “UPA Interface describes the interface to the UPA. Section 7.3.4, “Other Interfaces described the signal interface to other blocks that do not fit the generic interface model. These blocks mainly include other control blocks.

Note: This section does NOT show all the interface signals to a block. but rather, only shows the signals that the Bus Controller is involved in. A block may have signals that are used to talk to other blocks which the Bus Controller does not see. These signals are not shown or described.

7.3.1 Generic Interface

The figure below shows the interface signals to a generic internal block.

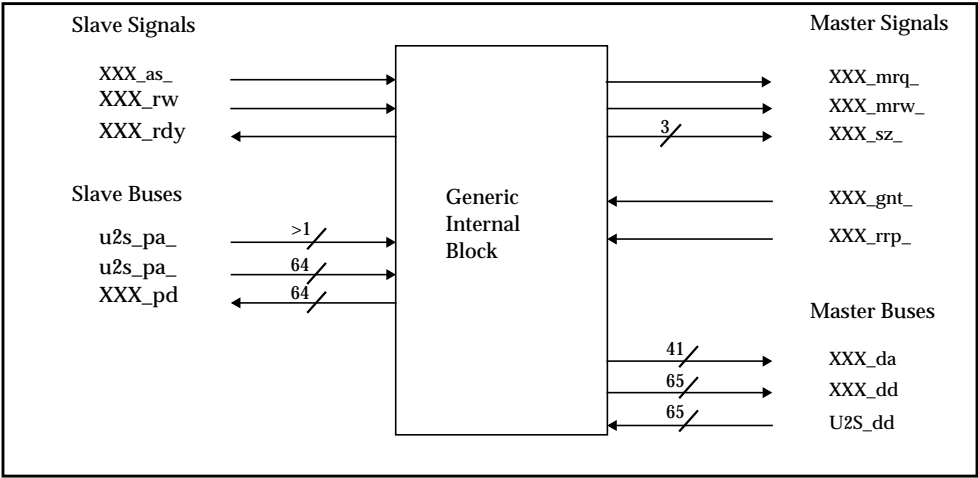


Figure 7-3 Generic Block Interface

Note: In the figure above and the following tables, the prefix XXX to each signal is replaced by the prefix of the internal block.

All blocks will have the slave signals for diagnostic reads and writes. Some blocks can act as a master, that is, the block can issue requests to the UPA. These blocks include the merge buffer, IOMMU, Mondo Dispatch, SBus Module, and Streaming Cache. These blocks will have the master signals.

Table 7-1 Slave Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| XXX_as_ | 1 | I | Address Strobe - when asserted, indicates a slave (PIO) access |
| XXX_rw | 1 | I | Read/Write - valid when XXX_as_ is asserted to indicate a slave (PIO) read to write |
| XXX_rdy_ | 1 | O | Ready - when asserted, indicates that the slave (PIO) access is complete |

Table 7-2 Master Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| XXX_mrqr_ | 1 | O | Master Request - when asserted, indicates that the block wishes to generate a request |
| XXX_mrwr | 1 | O | Master Read/Write - valid when XXX_mrqr_ is asserted to indicate a master read or write |
| XXX_sz | 2 | O | Size - valid when XXX_mrqr_ is asserted and indicates the size of the transfer (1,2,4,8,16,32, or 64 bytes) |
| XXX_gnt_ | 1 | I | Grant - when asserted, indicates that the block has won arbitration; the block should drive the address/data buses this cycle |
| XXX_rrp_ | 1 | I | Read Reply - when asserted, indicates that read reply data is valid on the input data bus |

7. Bus Controller

Table 7-3 Slave Buses

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| u2s_pa | Varies | I | UPA-to-SBus PIO Address - for Slave (PIO) accesses, specifies address to read or write |
| u2s_pd | 64 | I | UPA-to-SBus PIO Data - carries Slave (PIO) write data |
| XXX_pd | 64 | O | (Output) PIO Data - carries Slave (PIO) read data |

Table 7-4 Master Buses

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| XXX_da | Varies | O | (Output) DMA Address - for Master accesses, carries the address or header |
| XXX_dd | 65 | O | (Output) DMA Data - carries Master (DMA) write data; includes 1 bit to force bad ECC to the UPA |
| u2s_dd | 65 | I | UPA-to-SBus2 DMA Data - carries Master (DMA) read data; includes 1 bit of ECC uncorrectable error status |

7.3.1.1 Slave (PIO) Accesses

PIO reads or writes from the UPA to an internal block utilize the slave interface to each block. In both cases (reads and writes), the internal buses are frozen until the transaction completes. The intent is to keep the protocol as simple as possible. Generally, a PIO read or write to an internal block can be serviced immediately.

Only 64 bit (8 byte) PIO accesses are allowed to internal blocks.

PIO Read Cycles:

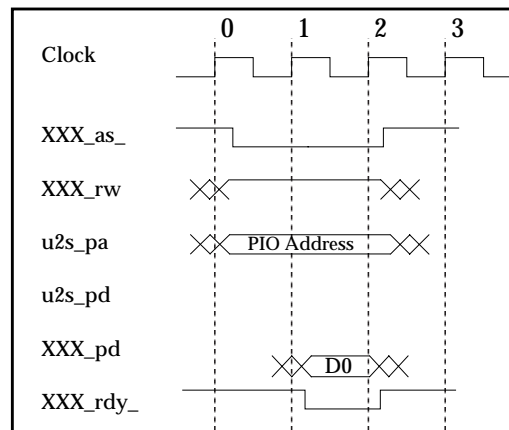


Figure 7-4 PIO Read to an Internal Block Timing

1. The XXX_as_ is asserted. The XXX_rw indicates a read. The u2s_pa indicates the address to be read. The u2s_pd and the XXX_pd are frozen.
2. The XXX_rdy_ is asserted. The XXX_pd is valid with the read data. The read is completed.

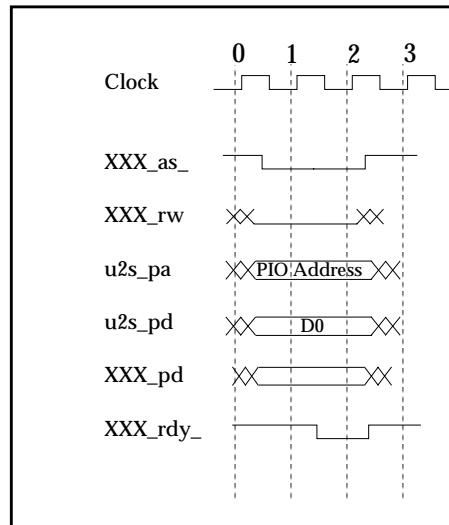
PIO Write Cycles:

Figure 7-5 PIO Write to an Internal Block Timing

1. The XXX_as_ is asserted. The XXX_rw indicates a write. The u2s_pa indicates the address to be written. The u2s_pd is valid with the write data. The u2s_pd and the XXX_pd are frozen.
2. The XXX_rdy_ is asserted to indicate that the block has taken the data. The write is completed.

7.3.1.2 Master Accesses

Certain blocks must have the capability of generating a request. Such blocks include the IOMMU for table walks, the Merge Buffer for writebacks, the Mondo Dispatch Unit for sending interrupts to the UPA, the Streaming Cache for issuing read prefetches and write flushes, and the SBus for issuing consistent DMA reads and writes. These blocks will use the master signals in participating in the arbitration sequence of the U2S.

When the block has a request it needs to issue to the UPA, it raises its request lines. Later, the Bus Controller will inform the block that it has won through the grant (XXX_gnt_) signal. The block must drive the address and data buses upon seeing the grant signal. It has a fixed period of time it is allowed to drive the buses which is based on the type of transfer it is performing.

Master Read Cycles:

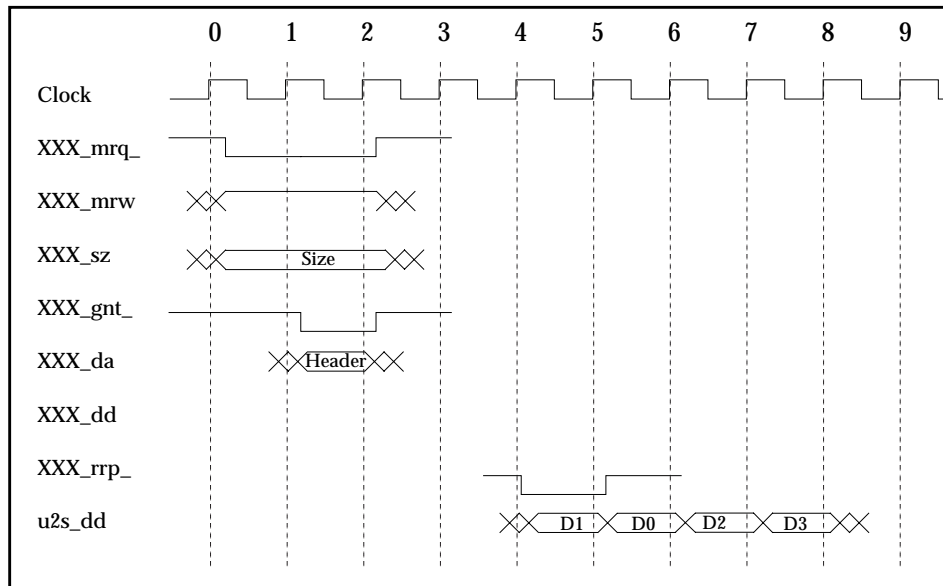


Figure 7-6 Master Read Timing

1. XXX_mrql_ is asserted. XXX_mrw indicates a read. XXX_sz indicates the size of the read.
2. Later, XXX_gnt_ is asserted by the Bus Controller to indicate that the block has won arbitration. The block will drive the address bus next cycle.
3. The block drives XXX_da. Note that the data bus is not used.
4. Later, XXX_rrp_ is asserted to indicate that the read data is coming back. s2u_dd is valid for consecutive cycles with the read reply data. Note that the above figure illustrates 4 cycles of data, this is not always the case.

Note: XXX_mrql_ for a request must be deasserted after XXX_gnt_ is asserted to indicate that it has won the arbitration, unless the block has another transfer to make.

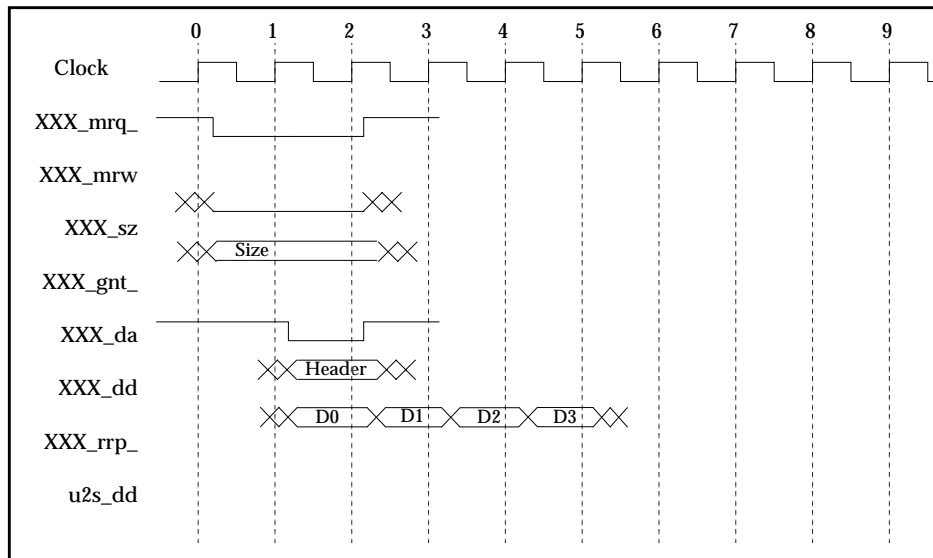
Master Write Cycles:

Figure 7-7 Master Write Timing

1. XXX_mrqr_ is asserted. XXX_mrwr indicates a write. XXX_sz indicates the size of the transfer.
2. Later, XXX_gnt_ is asserted by the bus controller to indicate that the block has won arbitration. The block should drive the buses the next cycle.
3. The block drives XXX_da with the address and XXX_dd with the data.

Note: XXX_mrqr_ for a request must be deasserted after XXX_gnt_ is asserted to indicate that it has won the arbitration, unless the block has another transfer to make.

7.3.1.3 DMA Read Reply Errors

Two types of DMA Read Reply Errors can occur:

- ECC uncorrectable error
- UPA time-out or other error

Each master block which issues DMA Reads must handle both error types. These blocks include the IOMMU, Merge Buffer, Streaming Cache, and SBus Module.

For ECC uncorrectable errors, there is a status bit associated with the u2s_dd bus indicated good or bad data for each transfer cycle. Otherwise ECC uncorrectable errors are transparent to the Bus Controller. Data will be delivered to the block as usual.

Other errors on the UPA, such as time-out errors, are dealt with in 2 different ways. First, for the IOMMU and Merge Buffer, each block will receive a Nack signal (mmu_tnk_ and dcc_rnk_) indicating that the transfer was not completed. The Bus Controller is not involved with this. Second, for the streaming cache and SBus Module, the return header is sent as usual with an error signal asserted (stc_err_ and sbm_err_). No data is sent.

7.3.2 Specific Interfaces

Each internal blocks have a slightly different interface than the generic case. Most of the signals are common, however, there are a few additions and subtractions to each block.

The table below lists the prefixes for each block.

Table 7-5 Block Prefixes

| Prefix | Block |
|--------|------------------------------|
| mmu | IOMMU |
| mdu | Mondo Dispatch Unit |
| stc | Streaming Cache |
| sbm | SBus Module |
| dcc | Merge Buffer (was DMA Cache) |
| ecc | ECC/Parity Unit |
| usr | UPA Slave Receiver |
| ust | UPA Slave Transmitter |
| umr | UPA Master Receiver |
| umt | UPA Master Transmitter |
| prf | Performance Counters |

For simplicity in verilog integration, the prefix for the DMA cache (DCC) was kept for the merge buffer.

7.3.2.1 IOMMU Interface

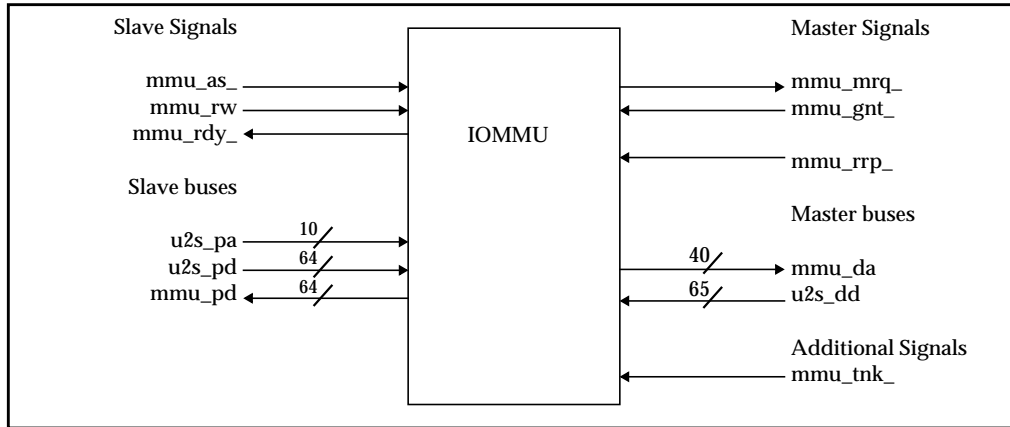


Figure 7-8 IOMMU Interface

Table 7-6 Additional IOMMU Signals

| Signal Name | Signals | I/O | Description |
|-----------------------|---------|-----|--|
| <code>mmu_tnk_</code> | 1 | I | IOMMU Table Walk Nack - when asserted, indicates an error has occurred for the Table Walk and no data will be returned |

The `mmu_tnk_` signal is required in case there is an error on the UPA other than an uncorrectable ECC error. The IOMMU must be able to continue and drop the table walk, since it is stalled while waiting for the table walk.

The signals `mmu_mr_` and `mmu_sz` have been removed because they are constant for the IOMMU. For master requests - for example, IOMMU Table walks - the IOMMU will always do a DMA read to memory of 64 Bytes.

The `mmu_da` signal is driven with 41 bits of the physical address for the read.

7.3.2.2 Mondo Dispatch Unit Interface

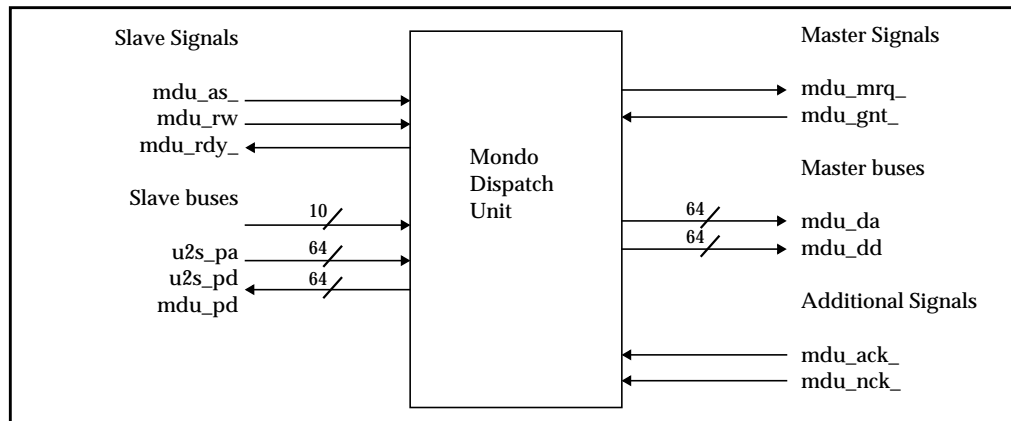


Figure 7-9 Mondo Dispatch Unit Interface

Table 7-7 Additional Mondo Dispatch Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| MDU_ACK_ | 1 | I | Mondo Dispatch Ack - asserted to indicate that the last interrupt sent to UPA has been Acked |
| MDU_NCK_ | 1 | I | Mondo Dispatch Nack - asserted to indicate that the last interrupt sent to UPA has been Nacked |

The signals `mdu_ack_` and `mdu_nck_` are handshakes from the UPA interface. These signals are used to inform the Mondo Dispatch Unit whether its last interrupt sent to the UPA has been processed.

As with the IOMMU, the `mdu_rw` and `mdu_sz` signals are removed since they are constant. The Mondo Dispatch Unit only issues a DMA Write of 64 Bytes.

7.3.2.3 DMA Merge Buffer Interface

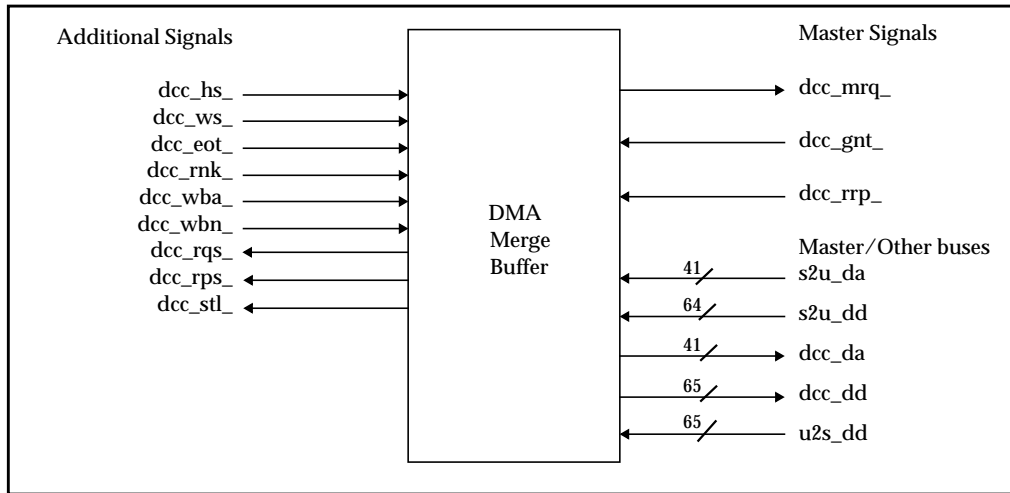


Figure 7-10 Merge Buffer Interface

Table 7-8 Additional Merge Buffer Signals

| Signal Name | Signals | I/O | Description |
|-----------------------|---------|-----|---|
| <code>dcc_hs_</code> | 1 | I | Merge Buffer Header Strobe - asserted to insert a DMA Write Header into the Merge Buffer |
| <code>dcc_ws_</code> | 1 | I | Merge Buffer Write Strobe - write enable signal for writing data into the Merge Buffer |
| <code>dcc_eot_</code> | 1 | I | Merge Buffer End of Transmission - asserted to indicate that all DMA Write data has been written into the Merge Buffer |
| <code>dcc_rnk_</code> | 1 | I | Merge Buffer Read Nack - when asserted, indicates an error on the UPA for the last Read-to-Own issued by the DCC; the DMA Write data will be lost |
| <code>dcc_wba_</code> | 1 | I | Merge Buffer Writeback Ack - when asserted, indicates that the Writeback has completed |
| <code>dcc_wbn_</code> | 1 | I | Merge Buffer Writeback Nack - when asserted, indicates that the Writeback has been canceled |
| <code>dcc_rqs_</code> | 1 | O | Merge Buffer Request Strobe - asserted to start a Writeback operation |

Table 7-8 Additional Merge Buffer Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| dcc_rps_ | 1 | O | Merge Buffer Reply Strobe - asserted to start a Read-to-Own Reply operation |
| dcc_stl_ | 1 | O | Merge Buffer Stall - when asserted, no more DMA transactions from any source should win arbitration |

Table 7-9 Additional DMA Cache buses

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| s2u_da | 41 | I | SBus2-to-UPA Address - carries the DMA Write Subline |
| s2u_dd | 64 | I | SBus2-to-UPA Data - carries the DMA Write data |

The Merge Buffer has no slave interface. No PIO is performed to this block.

The Merge Buffer processes one DMA transaction at a time. During processing, dcc_stl_ is asserted which indicates that no DMA transaction that requires the DCC must win arbitration. This is to preserve ordering within the U2S, since the DMA Write must be merged and written to memory before any other DMA transactions can be processed.

The signal dcc_rnk_ is asserted when the UPA has a time-out error or other such error that do not include an uncorrectable ECC error (which is carried with the returning data). In this case the Merge Buffer simply aborts the write-subline.

7.3.2.4 SBus Module

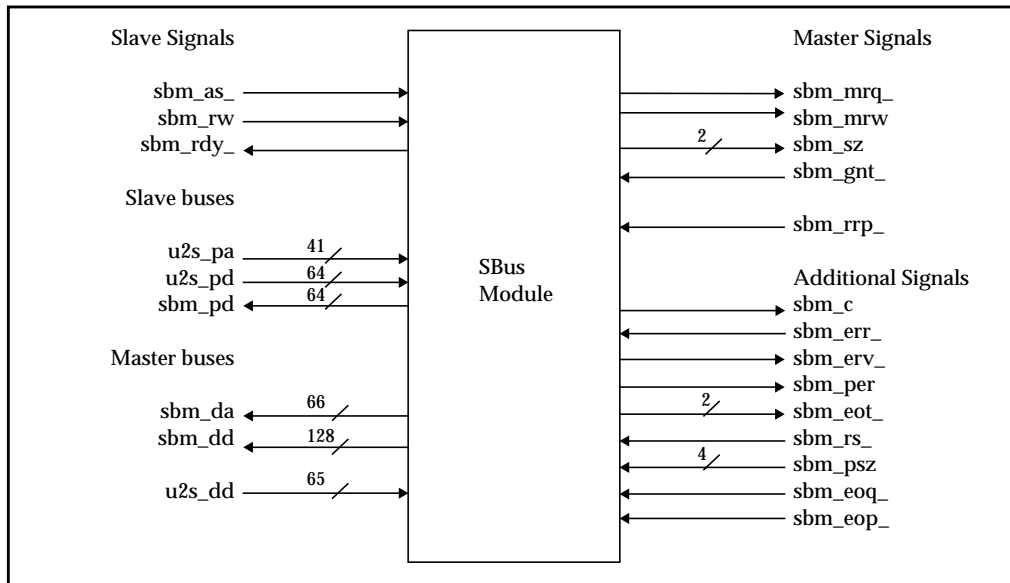


Figure 7-11 SBus Module Interface

Table 7-10 Additional SBus Module Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| sbm_c | 1 | O | SBus Module Cacheable Bit - valid when sbm_mr_ is asserted to indicate if the transaction is to memory or not |
| sbm_err_ | 1 | I | SBus Module Error - valid when sbm_rrp_ is asserted to indicate an error on a DMA Read on UPA that does not include an uncorrectable ECC error |
| sbm_erv_ | 1 | I | SBus Module Error Valid - asserted to indicate sbm_per is valid |
| sbm_per | 2 | O | SBus Module PIO Error - valid when sbm_rdy_ is asserted to indicate a PIO Read error on SBus (1 bit for time-out, 1 bit for any other error) |
| sbm_eot_ | 1 | O | SBus Module End of Transmission - asserted for PIO accesses to notify the Bus Controller ahead of time that the transfer is completed on SBus; coincides with the ACK on SBus |

Table 7-10 Additional SBus Module Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| sbm_rs_ | 1 | I | SBus Module Read Strobe - asserted when taking data out of the SBM for DMA Write Request |
| sbm_psz_ | 1 | I | SBus Module PIO Size - valid when sbm_as_ is asserted to indicate the size of the PIO transaction |
| sbm_eoq_ | 1 | I | SBus Module End-of-Request - asserted to indicate last cycle of DMA Write Request data taken |
| sbm_eop_ | 1 | I | SBus Module End-of-Reply - asserted to indicate last cycle of DMA Read Data inserted into the SBM |

The signal sbm_eot_ is used for PIO Read replies from SBus. It is asserted one cycle after the last ACK on SBus. This allows the Bus Controller to notify the UPA interface to send the P_REPLY early, saving latency (see timing diagrams). sbm_as_, sbm_rw, abm_psz, and u2s_pa will be invalid 1 cycle following sbm_eot.

PIO accesses to internal blocks are only 1 cycle on the internal buses (64 bits). However, a PIO access to SBus can be multiple cycles. The sbm_rdy_ signal is asserted every cycle data is transferred. So for PIO Writes, sbm_rdy_ indicates that the SBM has taken the data and the next cycle of data should be driven. For PIO Reads, every cycle sbm_rdy_ is asserted indicates that the SBM is driving valid data on the bus that cycle.

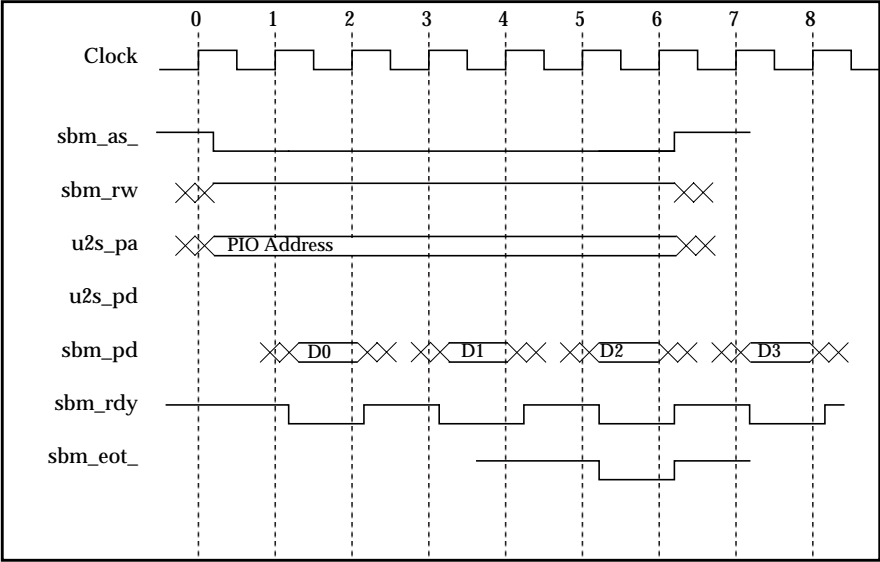


Figure 7-12 PIO Read to SBus Timing

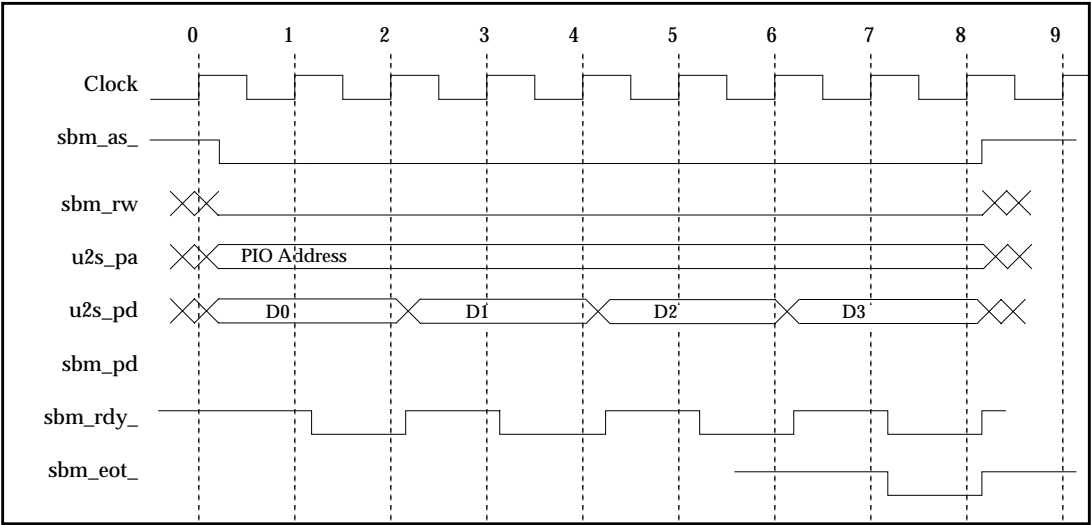


Figure 7-13 PIO Write to SBus Timing

DMA from/to SBus is slightly different. For DMA Reads, when the data returns from the UPA, sbm_rrp_ is asserted for the duration of the transfer (4 cycles, see Figure 7-14 and Figure 7-15), then sbm_eop_ is asserted the last transfer cycle.

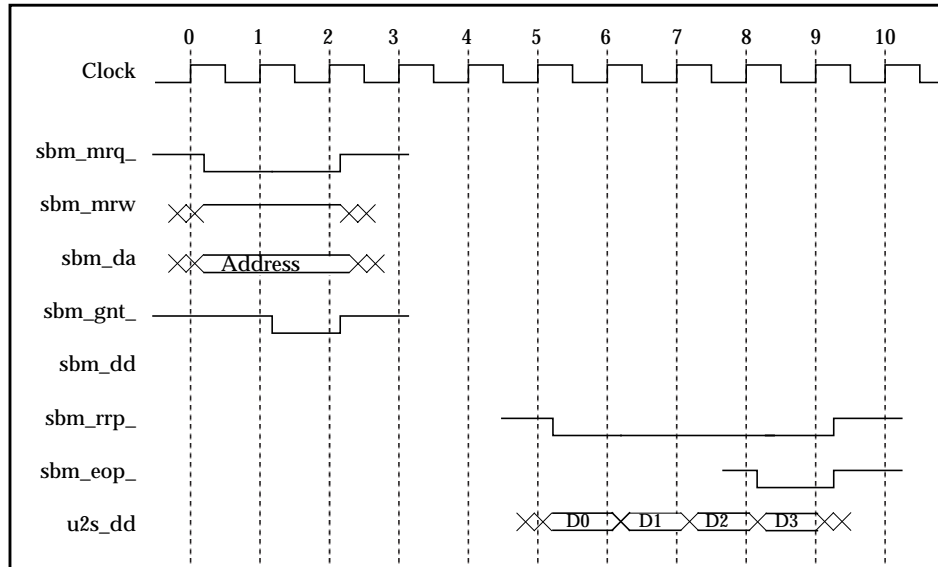


Figure 7-14 SBus DMA Read Timing

For DMA Writes, sbm_rs_ is asserted for the duration of the transfer, and sbm_eoq_ is asserted the last cycle.

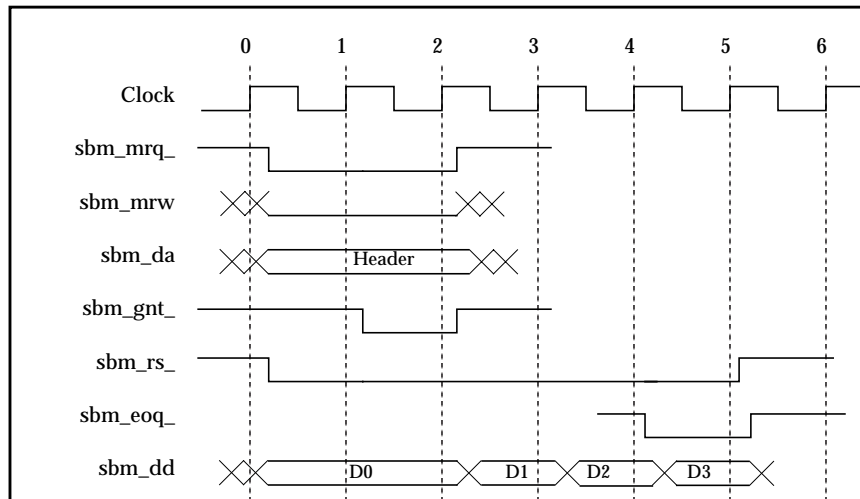


Figure 7-15 SBus DMA Write Timing

If sbm_err_ is asserted on a read reply, no data will be transferred.

The sbm_c signal is used during arbitration. If it equals 1, then the transaction goes to memory (cacheable bit set). If it equals 0, then the transaction goes to IO space or another UPA port (cacheable bit clear). The following table lists the transactions from the SBM.

Table 7-11 SBM DVMA Consistent Transactions

| Type | SBM_C | Size (Bytes) | Action |
|-------|-------|--------------|--|
| Read | 0 | 1 to 16 | Goes to the UPA as a single cycle read |
| Read | 0 | 32, 64 | Goes to the UPA as a four cycle read (64 bytes); if a 32 byte read, the extra 32 bytes will be thrown away |
| Read | 1 | Any | Goes to the UPA as a four cycle read; extra data will be thrown away |
| Write | 0 | 1 to 16 | Goes to the UPA as a single cycle write |
| Write | 0 | 32 | Goes to the UPA as two signal cycle writes |
| Write | 0 | 64 | Goes to the UPA as a four cycle write |
| Write | 1 | 1 to 32 | Goes to the Merge Buffer |
| Write | 1 | 64 | Goes to the UPA as a four cycle write |

7.3.2.5 Streaming Cache Interface

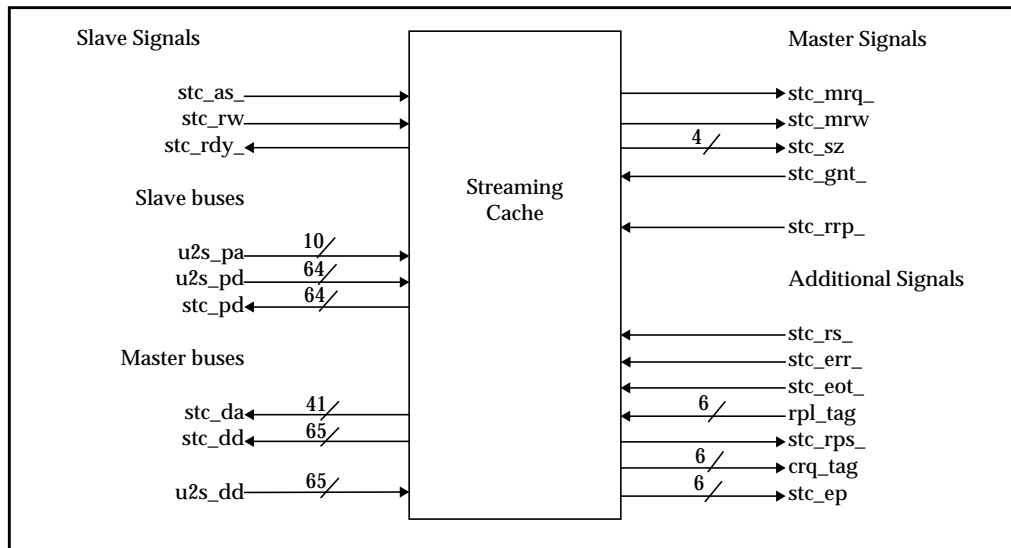


Figure 7-16 Streaming Cache Interface

Table 7-12 Additional Streaming Cache Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| stc_err_ | 1 | I | Streaming Cache Error - valid when stc_rrp_ is asserted to indicate an error on UPA for a DMA Read |
| stc_rs_ | 1 | I | Streaming Cache Read Strobe - asserted each cycle to take the data out of the streaming cache for a DMA Write |
| stc_eot_ | 1 | I | Streaming Cache End-of-Transmission - asserted on the last cycle of Read Reply data |
| rpl_tag | 6 | I | Reply Tag - 6 bit reply tag valid for DMA Read Replies |
| stc_rps_ | 1 | O | Streaming Cache Reply Strobe - asserted after stc_rrp_ to inform the Bus Controller to start transmitting DMA Read Reply data |
| crq_tag | 6 | O | Streaming Cache Request Tag - 6 bit tag that will be returned when the reply comes back |
| stc_ep | 6 | O | Streaming Cache End Pointer - points to the byte after the last valid byte of a DMA Write |

The `stc_sz` has only three values for the Streaming Cache (for DMA Write). If `stc_sz` equals 4 bytes (010), then only 1 data cycle is transferred to the DMA Merge Buffer. This is a special descriptor update. If `stc_sz` equals 64 bytes (110), then all valid 64 bytes of data goes out to the UPA as a Write-Invalidate. Any other value that `stc_sz` has is treated as a line flush in which 8 cycles on the data bus is transferred to the Merge Buffer but only a some of the data is valid. In this case, the 6 LSBs of `sbm_da` indicates the starting byte of the valid data, and `stc_ep` indicates the ending byte of the valid data.

The `stc_err_` will be asserted for an error condition on UPA that does not include an uncorrectable ECC error. No data will be transferred, only the reply tag (`rpl_tag`).

7.3.2.6 ECC and Parity Interface

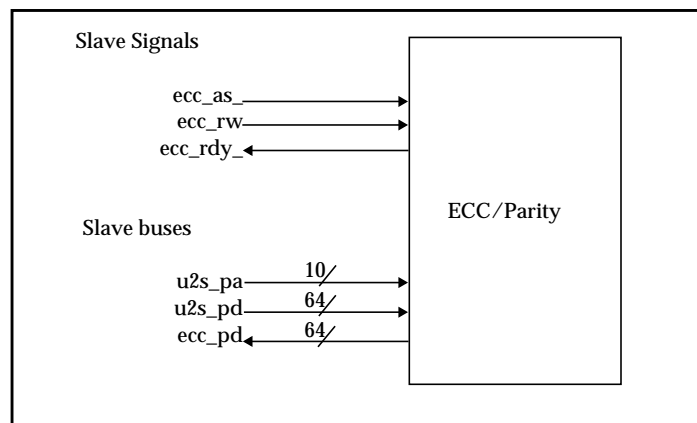


Figure 7-17 ECC/Parity Interface

7.3.2.7 Timer Counter Interface

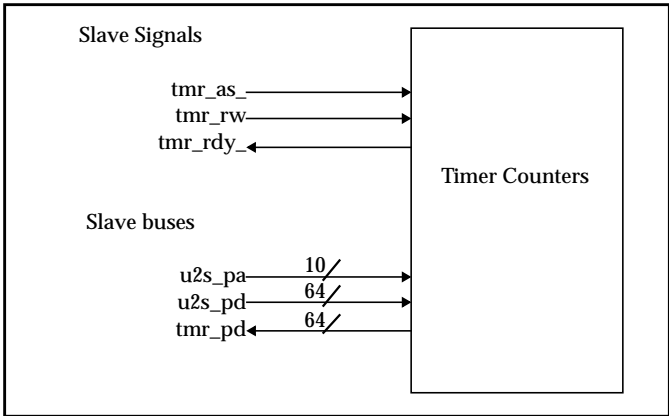


Figure 7-18 Timer Counter Interface

7.3.2.8 Performance Monitors Interface

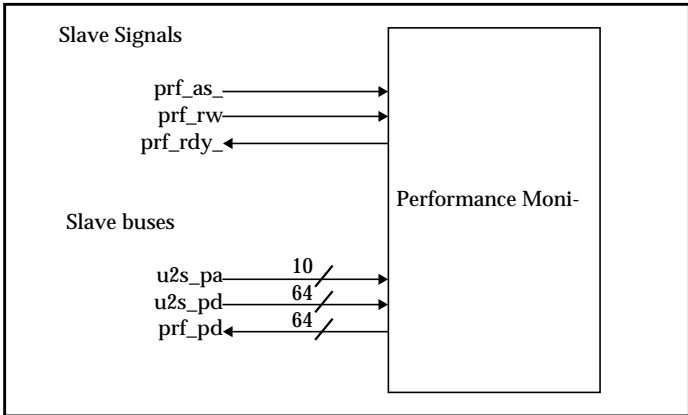


Figure 7-19 Performance Monitor Interface

7.3.3 UPA Interface

The main objective to the UPA interface is to keep it as consistent with the other blocks as possible. Thus, many of the generic interface signals are used (for example: XXX_mrql_, XXX_sz). Although the interface signals to the Bus Controller appear to be coming from each UPA block, the architecture of the UPA causes the signals to be physically originating from different sources in the U2S. Though they can be thought of as logically being a part of the UPA interface.

One example of this is the size encoding of a UPA Read Reply. A UPA Read Reply does not have any header associated with it; so when the data comes back, the UPA block cannot supply XXX_sz. It is the DMA Scoreboard that keeps the original transaction information. So, the DMA Scoreboard is the block that actually generates XXX_sz for the UPA block. To the Bus Controller, the UPA appears to be one logical block; however, the signals are physically coming from multiple blocks.

7.3.3.1 UPA Slave Receiver Interface

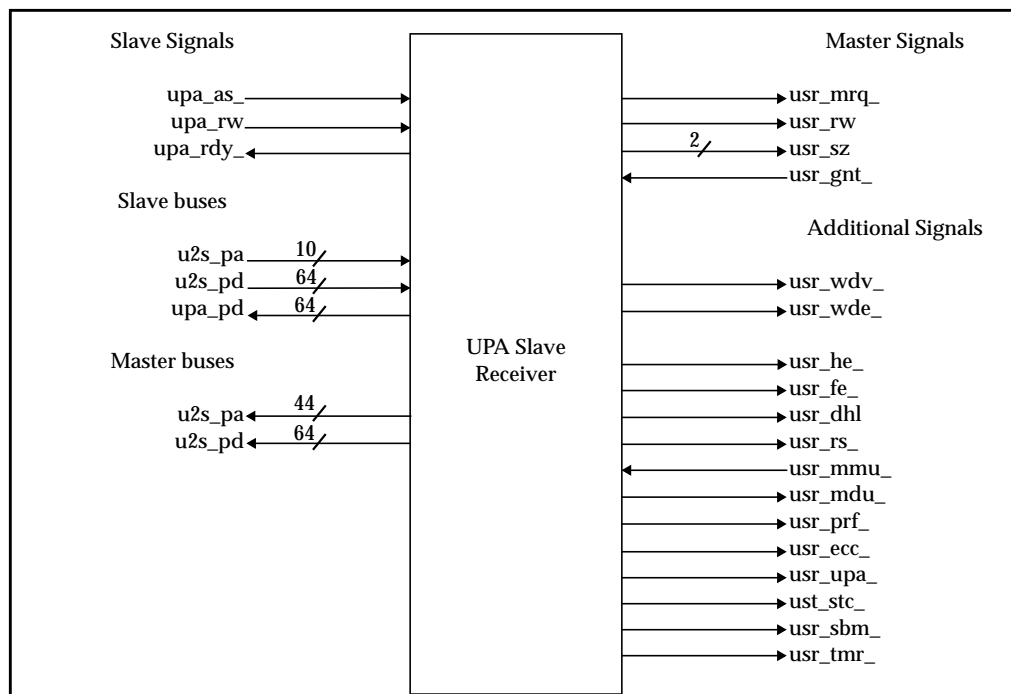


Figure 7-20 UPA Slave Receiver Interface

The UPA Slave Receiver block is responsible for handling PIO (Slave) requests to the U2S.

Table 7-13 Additional UPA Slave Receiver Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| usr_he_ | 1 | O | UPA Slave Receiver Header Error - valid when usr_mr_ is asserted to indicate a parity error in the UPA header |
| usr_fe_ | 1 | O | UPA Slave Receiver Format Error - valid when usr_mr_ is asserted to indicate a invalid UPA header, invalid size, or invalid transaction |
| usr_wdv_ | 1 | O | UPA Slave Write Data Valid - asserted to indicate valid PIO Write data |
| usr_wde_ | 1 | O | UPA Slave Write Data Error - asserted to indicate PIO Write data with uncorrectable ECC |
| usr_dhl | 1 | O | UPA Slave Receiver Data High/Low - valid when usr_mr_ is asserted and the size is 8 bytes or less on a write; indicates where the valid data is located |
| usr_rs_ | 1 | I | UPA Slave Receiver Read Strobe - asserted to advance the 128 bits FIFO; indicates that the PIO write data has been taken |
| usr_mmu_ | 1 | O | UPA Slave Receiver to IOMMU - asserted to indicate that the IOMMU is the target of the PIO |
| usr_mdu_ | 1 | O | UPA Slave Receiver to Mondo - asserted to indicate that the Mondo Dispatch Unit is the target of the PIO |
| usr_prf_ | 1 | O | UPA Slave Receiver to Performance Monitors - asserted to indicate that the Performance Monitors is the target of the PIO |
| usr_ecc_ | 1 | O | UPA Slave Receiver to ECC - asserted to indicate that the ECC is the target of the PIO |
| usr_upa_ | 1 | O | UPA Slave Receiver to UPA - asserted to indicate that the UPA is the target of the PIO |
| usr_stc_ | 1 | O | UPA Slave Receiver to Streaming Cache - asserted to indicate that the Streaming Cache is the target of the PIO |
| usr_sbm_ | 1 | O | UPA Slave Receiver to SBus Module - asserted to indicate that the SBus Module is the target of the PIO |

Table 7-13 Additional UPA Slave Receiver Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| usr_tmr_ | 1 | O | UPA Slave Receiver to Timer Counters - asserted to indicate that the Timer Counters are the target of the PIO |

The UPA Slave Receiver Interface is the logical interface that the Bus Controller sees from the UPA for PIO Accesses. However, some of the signals come from the PIO Decoder, which is logically associated with the UPA Slave Receiver. The partitioning is shown in Figure 7-21, "UPA Slave Receiver Signal Partition."

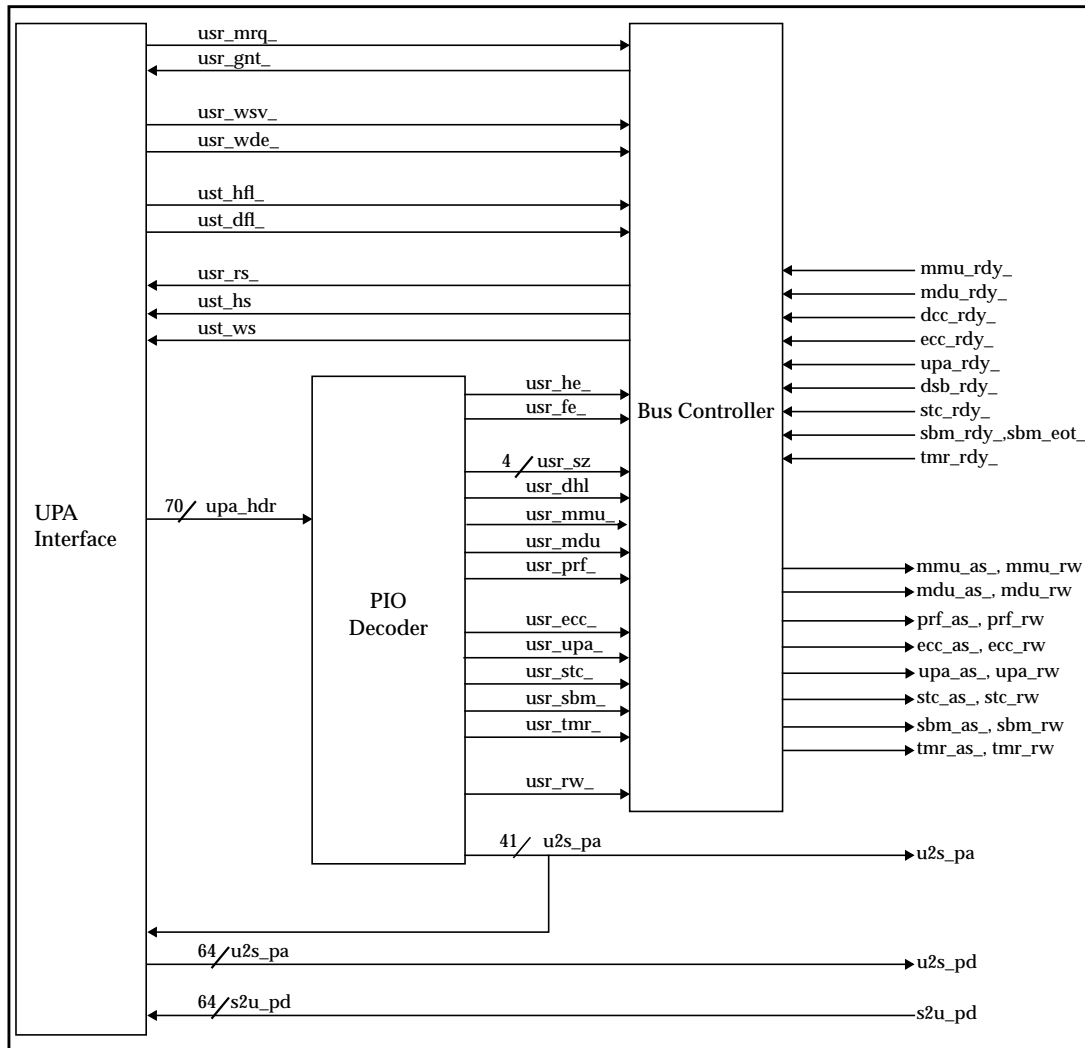


Figure 7-21 UPA Slave Receiver Signal Partition

The Bus Controller begins the PIO access when `usr_mrq_` is asserted. The `usr_gnt_` informs the UPA to advance its PIO header FIFO.

If `usr_he_` or `usr_fe_` is asserted when `usr_mrq_` is asserted, then the header has a PIO error. In this case, the Bus Controller will clock the transaction out to the UPA and inform the P_REPLY unit.

For PIO Writes, the Bus Controller, upon seeing `usr_mrq_`, waits for `usr_wdv_` to be asserted to indicate that the PIO Write data has arrived before proceeding with the PIO access. If `usr_wde_` is asserted, then the write data has an uncorrectable ECC error. The Bus Controller will clock the write out of the FIFOs (including data).

For PIO Reads, `usr_rs_` is used to advance the UPA PIO Read FIFO.

7.3.3.2 UPA Slave Transmitter Interface

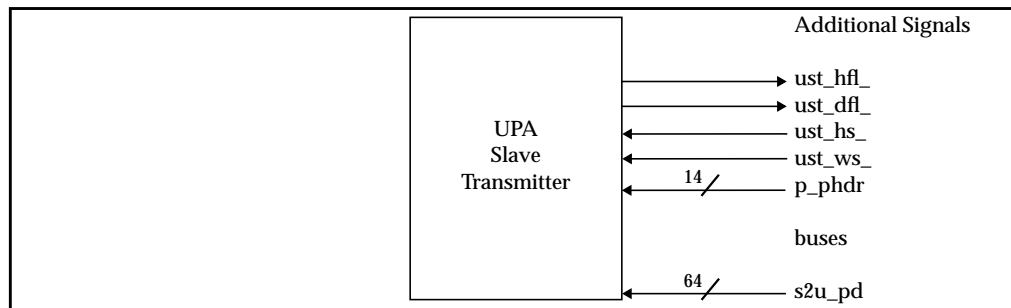


Figure 7-22 UPA Slave Transmitter Interface

The UPA Slave Transmitter is responsible for handling PIO replies.

Instead of a returning a header, the UPA requires P_REPLY. The information needed to generate P_REPLY is given by `p_phdr`, which is valid when `ust_hs_` is asserted. The components of `p_phdr` is given below.

Table 7-14 `p_phdr`

| Signal Name | Signals | Description |
|---------------------------|---------|---|
| <code>p_phdr[13:9]</code> | 5 | MID - original MID of the PIO transaction |
| <code>p_phdr[8]</code> | 1 | Class - original class of the PIO transaction |
| <code>p_phdr[4:7]</code> | 4 | Transaction Type - UPA encoding |
| <code>p_phdr[3]</code> | 1 | Format Error - <code>usr_fe_</code> |
| <code>p_phdr[2]</code> | 1 | Header Parity Error - <code>usr_he_</code> |
| <code>p_phdr[1:0]</code> | 2 | SBus PIO Error - <code>sbm_per</code> |
| Total Signals | 14 | |

From these signals, the UPA Slave Transmitter can determine the appropriate P_REPLY to generate.

7.3.3.3 UPA Master Receiver Interface

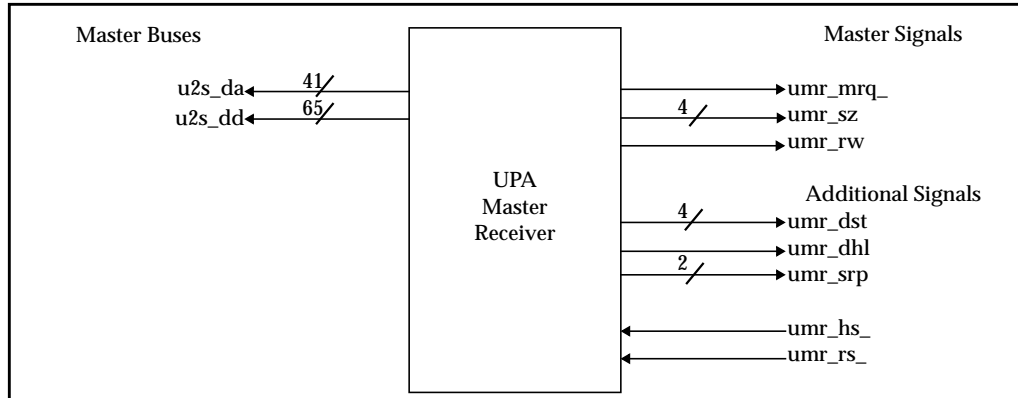


Figure 7-23 UPA Master Receiver Interface

The UPA Master Receiver is responsible for handling DMA read replies.

Table 7-15 Additional UPA Master Receiver Signals

| Signal Name | Signals | I/O | Description |
|-----------------------|---------|-----|---|
| <code>usr_dhl_</code> | 1 | O | UPA Data High/Low - valid when <code>usr_mrqr_</code> is asserted and when the size of the read is 8 bytes or less to indicate where the valid data is located (either in the upper 64 bits or lower) |
| <code>usr_dst</code> | 4 | O | UPA Master Receiver Destination - indicates the destination of the DMA Read Reply |
| <code>usr_srp</code> | 2 | O | UPA Master Receiver S_REPLY - encoded S_REPLY to indicate result of the DMA Read |
| <code>umr_rs_</code> | 1 | I | UPA Slave Receiver Read Strobe - asserted to advance the FIFO; indicates that the DMA Read data has been taken |
| <code>umr_hs_</code> | 1 | I | UPA Slave Receiver Header Strobe - used to advance the DMA Scoreboard |

Table 7-16 Destination/Source Encoding

| Code<3:0> | Destination/Source |
|-----------|----------------------|
| 0000 | |
| 0001 | |
| 0010 | |
| 0011 | |
| 0100 | |
| 0101 | IOMMU Block |
| 0110 | Mondo Dispatch Block |
| 0111 | |
| 1000 | Streaming Cache |
| 1001 | Merge Buffer |
| 1010 | |
| 1011 | |
| 1100 | SBus Module |
| 1101 | |
| 1110 | |
| 1111 | |

A reply from UPA does not have any header information. So the reply header must be generated by the DMA scoreboard, which stores the original transaction.

Table 7-17 UPA Interface Signals to DMA Controller Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| upa_rpl_ | 1 | O | UPA Reply - asserted to indicate a read or write reply |
| upa_rpa_ | 1 | I | UPA Reply Ack - asserted to acknowledge a upa_rpl_ |

7.3.3.4 UPA Master Transmitter

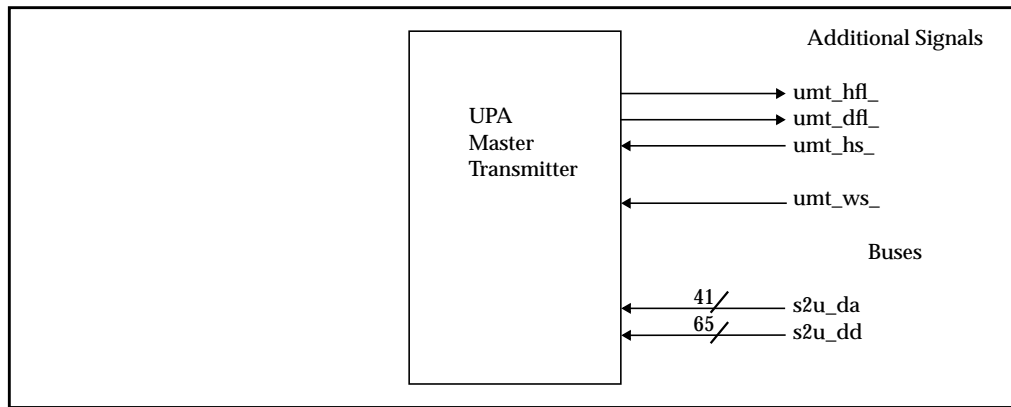


Figure 7-24 UPA Master Transmitter Interface

The UPA Master Transmitter is responsible for handling DMA and Interrupt Requests to the UPA.

When the extra line in `s2u_dd` is asserted, a bad ECC should be generated for the data.

7.3.4 Other Interfaces

This section describes interfaces to blocks that do not follow the generic interface.

7.3.4.1 DMA Scoreboard Interface

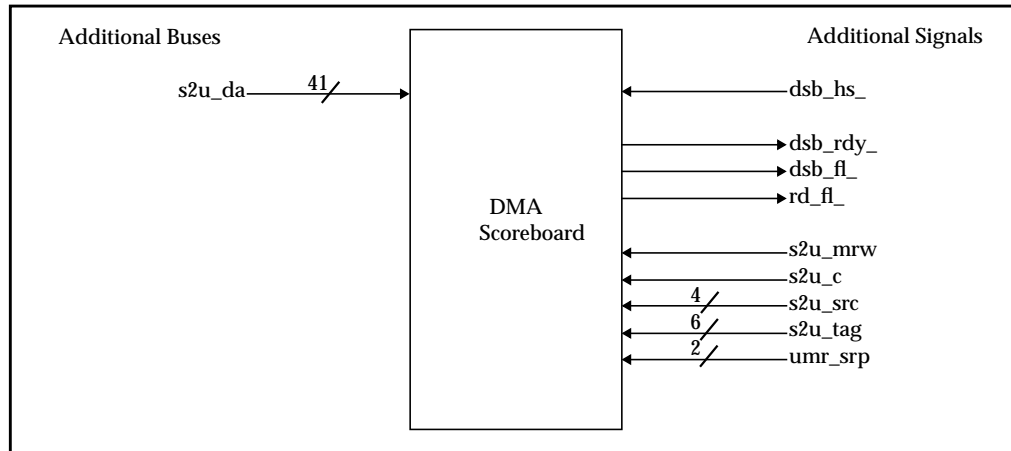


Figure 7-25 DMA Controller Interface

Table 7-18 Additional DMA Controller Signals

| Signal Name | Signals | I/O | Description |
|-----------------------|---------|-----|--|
| <code>dsb_rdy_</code> | 1 | O | DMA Scoreboard Ready - asserted when the DMA Scoreboard is ready to process a DMA Request |
| <code>dsb_hs_</code> | 1 | I | DMA Header Strobe - asserted when a DMA Request is on the <code>s2u_da</code> Bus |
| <code>dsb_dl_</code> | 1 | O | DMA Scoreboard Full - asserted when the scoreboard is full |
| <code>rd_fl_</code> | 1 | O | Read Full - asserted to indicate that no more DMA reads should be issued because the limit of the DMA Read FIFO has been reached |

Table 7-19 Additional DMA Controller buses

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| s2u_da | 41 | I | SBus2-to-UPA Address - carries the DMA Request Address to be logged into the scoreboard |

There is no slave interface to the DMA scoreboard. There are no PIO accesses to the DMA scoreboard.

All DMA transactions are logged into the DMA scoreboard. When these transactions win arbitration, in addition to being sent to their respective destinations, they are copied to the DMA Controller.

The `dsb_fl_` is asserted when the DMA scoreboard is full. In these cases, no DMA transaction can win arbitration until `dsb_fl_` signal is de-asserted (for example: replies have come back to clear entries in the scoreboard).

See the UPA Master Receiver for other DMA Controller signals that go to the Bus Controller.

7.4 Bus Controller Functional Description

The following section is a discussion of each Bus Controller, their responsibilities, which blocks they interact with, and the actions they take.

Each controller (U2S, S2U, and PIO) controls a set of buses and is capable of transferring one transaction at a time. However, each is independent, so a maximum of three transactions can be transferred at any time (one transaction over each set of buses).

Each set of buses includes an address bus and a corresponding data bus. Headers will be transferred over the address bus and always take one cycle. Data will take one to eight cycles, depending on the size of the data. In some cases, depending on the transaction, only the address bus may be used, at which time the data bus corresponding to that address bus will be idle. For example, a DMA Read request is issued by the SBus Module. During this transfer, the s2u_da bus will be used to transfer the header for the read, but the s2u_dd bus will be idle for that cycle. In other cases, both address and data buses will be used. However, the data transfer could take multiple cycles, in which case, the address bus will be idle.

7.4.1 U2S Bus Controller

The U2S Bus Controller is responsible for transferring DMA Read Replies from the UPA (specifically the UPA Master Receiver) to the requesting U2S block. It controls the u2s_da and u2s_dd buses. The u2s_da Bus (DMA Address), carries header information back to the Streaming Cache and to the ECC unit, in case of an ECC error. The u2s_dd Bus (DMA Data) carries the Read Reply data. It is 64 bits wide.

Blocks which can receive Read Replies include the IOMMU, DMA Cache, Streaming Cache, and SBus (see following table for detailed descriptions). Reads issued from the IOMMU, DMA Cache, and Streaming Cache will be 64 bytes to the UPA. The IOMMU needs only 16 bytes, so the extra 48 bytes will be ignored. All Reads to memory from the SBus will also be 64 bytes (the extra data will be ignored). Reads to IO Space from SBus will be either 16 or 64 bytes (one or four UPA cycles). 32 Byte Reads to IO Space will be sent out as a 64 byte read with the extra data ignored.

DMA Write Replies are not handled by the U2S Bus Controller. Rather, the UPA Master Receiver will inform the DMA Controller directly, so that the DMA Scoreboard can be advanced. No address or data is transferred for a DMA Write Reply.

Table 7-20 U2S Bus Controller Actions

| Source | Transaction | Errors | Action (cycles) |
|--------------|-------------------|---------------------|--|
| IOMMU | Table Walk Reply | None | 1) mmu_rrp_ asserted; on the same cycle, 8 bytes of data is transferred; no address is transferred 2) 8 byte of data transferred; transfer complete 3) to 8) UPA FIFO is emptied of extra data |
| | | UE ECC Error | Same cycles as in the no error case - u2s_dd[64] contains the UE ECC Error indication; IOMMU will have to detect this and drop the data |
| | | UPA Error (no data) | No action taken - DMA Controller will notify the IOMMU through mmu_tnk_ signal |
| Merge Buffer | Read-to-Own Reply | None | 1) dcc_rrp_ asserted; the Bus Controller waits for dcc_rps_ 2) Later, dcc_rps_ is asserted by the Merge Buffer to indicate that it is ready; on the same cycle, 8 bytes of data is transferred; no address is transferred 3) to 9) 8 bytes transferred |
| | | UE ECC Error | Same cycles as in the no error case - u2s_dd[64] contains the UE ECC Error indication; the Merge Buffer loop this around to the ECC unit during Write-back and the ECC unit will generate bad ECC |
| | | UPA Error (no data) | No action taken - DMA Controller will notify the Merge Buffer through dcc_rnk_ signal |

Table 7-20 U2S Bus Controller Actions

| Source | Transaction | Errors | Action (cycles) |
|-----------------|---------------------|---------------------|---|
| Streaming Cache | Read Prefetch Reply | None | 1) stc_rrp_ asserted; the Bus Controller waits for stc_rps_ 2) Later, stc_rps_ is asserted by the Streaming Cache to indicate that it is ready; on the same cycle, 8 bytes of data is transferred, and the original index is returned on the address bus 3) to 9) 8 bytes transferred; on the last cycle stc_eot_ is asserted |
| | | UE ECC Error | Same cycles as in the no error case - u2s_dd[64] contains the UE ECC Error indication; the Streaming Cache will store the bad data and the error; when SBus requests the line, the Streaming Cache will indicate bad data |
| | | UPA Error (no data) | 1) stc_rrp_ and stc_err_ asserted; the Bus Controller waits for stc_rrp_ 2) Later, stc_rps_ is asserted by the Streaming Cache to indicate that it is ready; on the same cycle, the original index is returned on the address bus; no data is transferred |
| SBus Module | DMA Read Reply | None | 1) sbm_rrp_ asserted; 8 bytes transferred 2) to 7) 8 bytes transferred; sbm_eop_ asserted on last cycle |
| | | UE ECC Error | Same cycles as in the no error case - u2s_dd[64] contains the UE ECC Error indication; SBus will indicate an error during the transfer |
| | | UPA Error (no data) | 1) sbm_rrp_ and sbm_err_ asserted |

7.4.2 S2U Bus Controller

The S2U Bus Controller is responsible for making the transfers for DMA and Interrupt Requests out to the UPA and partial writes to the Merge Buffer. It controls the s2u_da (DMA Address) and s2u_dd (DMA Data) buses. The s2u_da bus carries the address and header information to the UPA Master Transmitter. The s2u_dd bus carries DMA Write data. It is 64 bits wide.

Blocks which can issue DMA Writes include the Mondo Dispatch Unit, Merge Buffer, Streaming Cache, and SBus (see following table for detailed descriptions). Writes issued from the Mondo Dispatch and Merge Buffer will be 64 bytes to the UPA. Writes from the Streaming Cache will also be 64 bytes; however, it is possible for only part of the data to be valid (for example a 64 byte line flush). If all 64 bytes are valid, then the write goes out to the UPA, else it goes to the Merge Buffer as a partial write. 64 byte writes to memory from the SBus will go out to the UPA. Less than 64 byte writes to memory from the SBus will go to the Merge Buffer. All writes to IO Space from SBus will go the UPA. Note that 32 byte writes to IO Space will be broken down as two 16 byte writes.

Table 7-21 S2U Bus Controller Actions

| Source | Transaction | Size | Destination | Action (cycles) |
|----------------|------------------|-----------------------------------|-------------|---|
| IOMMU | Table Walk | 16 Bytes needed but 64 Bytes Read | UPA | 1) mmu_mrq_ asserted by the Mondo Unit 2) Later, mmu_gnt_ asserted by the Bus Controller; on the same cycle, the IOMMU Unit sends the address |
| Mondo Dispatch | Interrupt Packet | Always 64 Bytes | UPA | 1) mdu_mrq_ asserted by the Mondo Unit 2) Later, mdu_gnt_ asserted by the Bus Controller; on the same cycle, the Mondo Unit sends the address and 8 bytes of data 3) to 9) 8 byte of data transferred |

Table 7-21 S2U Bus Controller Actions

| Source | Transaction | Size | Destination | Action (cycles) |
|-----------------|-----------------|-------------------------|--------------|--|
| Merge Buffer | Writebacks | Always 64 Bytes | UPA | 1) dcc_mrql_ asserted 2) Later, dcc_gnt_ asserted by the Bus Controller; the Bus Controller now waits for dcc_rqs_ 3) Later dcc_rqs_ asserted by the Merge Buffer; on the same cycle, the Merge Buffer sends the address and 8 bytes of data 3 to 9) 8 byte of data transferred |
| Streaming Cache | Writebacks | 64 Bytes of Valid Data | UPA | 1) stc_mrql_ asserted, stc_mrw = 0, stc_sz = 64 bytes by the Streaming Cache 2) Later, stc_gnt_ asserted by the Bus Controller; on the same cycle, the stc_rs_ asserted by the Bus Controller to read 8 bytes of data out, the Streaming Cache sends the address 3) to 9) 8 byte of data transferred |
| | Writebacks | <64 Bytes of Valid Data | Merge Buffer | 1) stc_mrql_ asserted, stc_mrw = 0 stc_sz < 64 bytes by the Streaming Cache 2) Later, stc_gnt_ asserted by the Bus Controller; on the same cycle, the stc_rs_ asserted by the Bus Controller to read 8 bytes of data out, the Streaming Cache sends the address 3) to 9) 8 byte of data transferred |
| | Read Prefetches | Always 64 Bytes | UPA | 1) stc_mrql_ asserted, stc_mrw = 1 by the Streaming Cache 2) Later, stc_gnt_ asserted by the Bus Controller; on the same cycle, the Streaming Cache sends the address |

Table 7-21 S2U Bus Controller Actions

| Source | Transaction | Size | Destination | Action (cycles) |
|-------------|-------------|--|--------------|--|
| SBus Module | DMA Write | 64 Bytes to Memory or any size to IO Space | UPA | 1) sbm_mrql_ asserted, sbm_mrql = 0 by the SBM 2) Later, sbm_gnt_ asserted by the Bus Controller; on the same cycle, the sbm_rs_ asserted by the Bus Controller to read 8 bytes of data out, the SBM sends the address 3) to 9) 8 byte of data transferred; sbm_eoql_ asserted on last cycle |
| | DMA Write | <64 Bytes to Memory | Merge Buffer | 1) sbm_mrql_ asserted, sbm_mrql = 0 by the SBM 2) Later, sbm_gnt_ asserted by the Bus Controller; on the same cycle, the sbm_rs_ asserted by the Bus Controller to read 8 bytes of data out, the SBM sends the address 3) to 9) 8 byte of data transferred; sbm_eoql_ asserted on last cycle |
| | DMA Read | Any size - to memory, goes out as 64 Bytes | UPA | 1) sbm_mrql_ asserted, sbm_mrql = 0 by the SBM 2) Later, sbm_gnt_ asserted by the Bus Controller; on the same cycle, the SBM sends the address |

7.4.3 PIO Bus Controller

The PIO Bus Controller is responsible for transferring PIO Read/Write and Copy-back/Invalidate Requests and Replies. It controls the u2s_pa bus (PIO Address), the u2s_pd Bus (PIO Data), and the s2u_pd Bus (PIO Data). Note that it controls the buses on both direction (unlike the S2U and U2S Bus Controllers). This is because for PIO transactions, the PIO Bus Controller freezes the buses in both directions until the transfer is complete. The u2s_pa bus is looped back around to the UPA Slave Transmitter to form the appropriate P_REPLY. Both PIO Data buses are 64 bits wide to all blocks.

Only 64 bit PIO accesses are support to internal blocks. However, any size transfer (that is support by the UPA) can go out to the SBus. The following table describes the PIO Bus Controller actions.

Table 7-22 PIO Bus Controller Actions

| Destination | Transaction | Size | Errors | Action (cycles) |
|-------------|-------------|-----------|--------|--|
| Any | PIO Read | 8 Bytes | None | 1) Appropriate XXX_as_ is asserted by the PIO Controller; the address is driven 2) Later, XXX_rdy_ is asserted by the block to indicate that the read data is available; 8 bytes are transferred into the UPA |
| | PIO Write | 8 Bytes | None | 1) Appropriate XXX_as_ is asserted by the PIO Controller; the address is driven; the data is driven 2) Later, XXX_rdy_ is asserted by the block to indicate that the data is taken |
| SBus | PIO Read | > 8 Bytes | None | 1) sbm_as_ is asserted by the PIO Controller; the address is driven 2) Later, sbm_rdy_ is asserted by the block to indicate that the read data is available; 8 bytes are transferred into the UPA step 2) is repeated until all data is read |

Table 7-22 PIO Bus Controller Actions

| Destination | Transaction | Size | Errors | Action (cycles) |
|-------------|-------------|-----------|--------------|--|
| | PIO Write | > 8 Bytes | None | 1) sbm_as_ is asserted by the PIO Controller; the address is driven; the data is driven 2) Later, sbm_rdy_ is asserted by the block to indicate that the data is taken step 2) is repeated until all data is written |
| Any | PIO Write | Any | UE ECC Error | 1) Bus Controller sees the UE ECC Error 2) to 9) 8 bytes are clocked out of the UPA FIFO; no data is transferred |
| Any | Any | Any | Parity Error | 1) Bus Controller sees the Parity Error 2) The P_REPLY Unit is notified; no data is transferred |

8.1 Definition of Terms

Abbreviations, TLAs and ETLAs used in this section:

- U2S: I/O Controller Chip
- UMS: UPA Master / Slave Control
- RMW: Read-Modify-Write
- SC: System Controller ASIC
- DVMA: Direct Virtual Memory Access Sun's version of DMA
- PIO: Programmed Input/Output
- UPA: Uniform Port Architecture
- S2U_A: SBus2 to UPA Address, used for all DVMA cycles
- U2S_A: UPA Address for PIO cycles plus system copyback requests
- UE: non-correctable error
- CE: Correctable error

8.2 UPA Master / Slave Overview

The UPA Master/Slave (UMS) block is the U2S's interface to the UPA_A request bus. The U2S can be addressed as a UPA Slave to handle PIO requests and can act as a UPA Master when it requires DMA transactions to be sent.

When the U2S is addressed as a UPA slave, all requests from the UPA address bus are written into the U2S_A FIFO. When the U2S wants to perform a DVMA request, the UMS arbitrates for use of the UPA Address bus and drives the request out from the S2U_A FIFO.

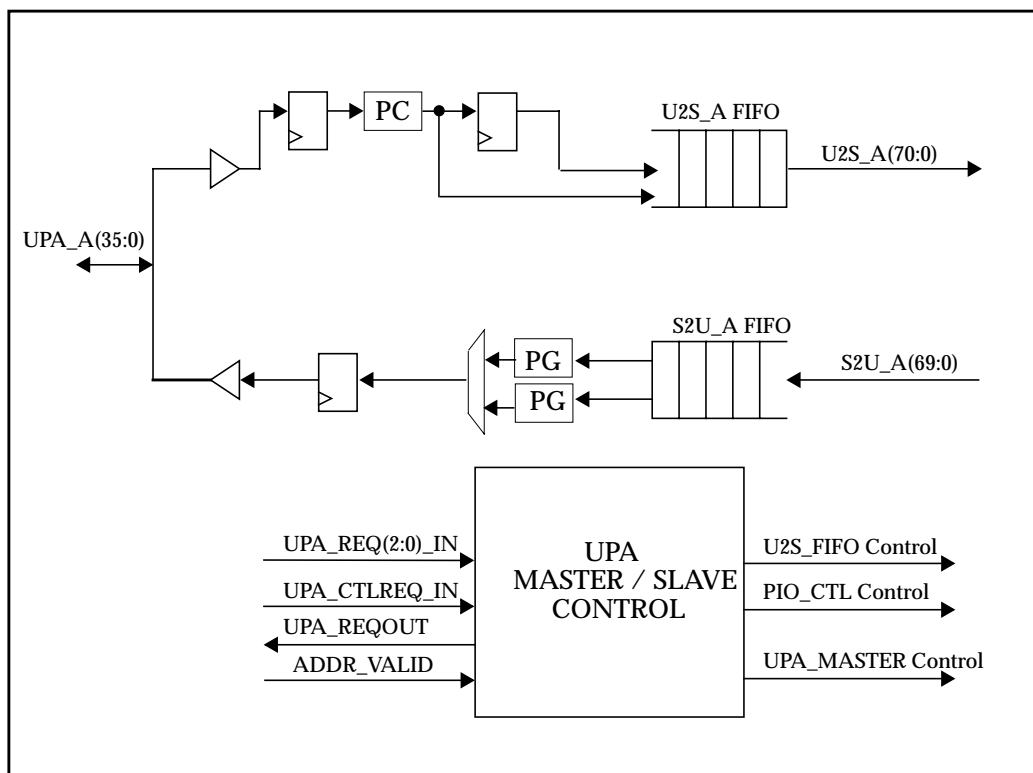


Figure 8-1 UPA Master / Slave High-level block diagram

8.3 Signal Descriptions

Table 8-1 Master/Slave Interface Signals

| Signal Name | Signals | I/O | Description |
|---------------|---------|-----|---|
| I_UPA_RQ | 3 | I | Request signals from other clients |
| I_SC_REQ | 1 | I | Request from system controller |
| I_ADDR_VLD | 1 | I | Indicates a valid address cycle on bus |
| I_UPA_A | 36 | I | UPA request bus |
| UPA_HDR | 71 | O | U2S_A request bus to the PIO controller unit |
| UPA_CLK_OUT | 1 | O | UPA_CLK_OUT (for debug purposes) |
| DMA_SRPLY | 1 | I | DMA related S_REPLY |
| O_UPA_REQ_OUT | 1 | O | Request from U2S indicating that it wants ownership of the UPA address bus (before the output register) |
| UPACLK | 1 | I | UPA clock |
| SIOCLK | 1 | I | U2S clock |
| UPARST_ | 1 | I | UPA Reset with respect to the UPA clock |
| SIORST_ | 1 | I | U2S Reset with respect to the U2S clock |
| I_ARBRST_ | 1 | I | Resets the Arbiter |
| USR_MRQ_ | 1 | O | Indicates the PIO controller that there is an entry in the U2S_A FIFO |
| O_ADDR_VLD | 1 | O | Address valid driven out on the UPA |
| ADDR_OE_B | 1 | O | Output Enable for driving Addr_Valid |
| UMT_HS_ | 1 | I | Enables the S2U_A FIFO to clock in the 64 bits of address from the DMA controller |
| UMT_HFL_ | 1 | O | Indicates that the S2U FIFO is full |
| U2S_PA | 2 | I | PIO Address to access a configuration register |
| U2S_PD | 64 | I | UPA to SBus PIO write data |
| UPA_PD | 64 | O | Output PIO read data |
| UPA_RW | 1 | I | Read/Write - valid when UPA_AS_ is asserted to indicate a PIO read or write |
| UPA_AS_ | 1 | I | Address strobe - when asserted indicates PIO access |

Table 8-1 Master/Slave Interface Signals

| Signal Name | Signals | I/O | Description |
|--------------|---------|-----|--|
| UPA_RDY_ | 1 | O | Ready - when asserted, indicates that the PIO access is complete |
| ADDR_LD_B | 1 | O | UPA Address bus load |
| ADDR_LD_D1B_ | 1 | O | Registered address_oe_nxt to load the input address buffers |
| MODE_BIT | 1 | O | Indicates whether the UPA clock is faster than the U2S clock |
| S2U_DA | 70 | I | DMA address bus from Bus Controller |
| SIO_IGN | 5 | O | Interrupt Group number |
| O_UPA_A | 36 | O | Output UPA Address bus |
| MID | 5 | O | UPA ID |
| U2S_RS | 1 | I | Advances the head pointer in the U2S FIFO |

8.4 UPA Master / Slave Functional Description

This block generates the control signals to the incoming and outgoing FIFOs from UPA_A bus.

If there was an incoming request in U2S_A FIFO the Slave logic would clock in the 64 bits of address from the bus in two consecutive cycles.

When there is an entry in the S2U_A FIFO, the UMS would arbitrate for the bus and if it becomes the winner, it then goes into the Master state. In this state it would send out the request on the UPA_A bus.

8.4.1 UPA Master (UM) Description

8.4.1.1 UM Overview

When there is an entry in the S2U_A FIFO it indicates that the U2S wants to send a request on the UPA Address bus. The arbitration block takes in the request and starts the arbitration for the address bus. The U2S will continue to arbitrate for the bus till it becomes the winner. If the U2S wins the bus then it becomes the current master. The control logic in the master then generates output enable to drive the output buffers. The state machine in the UM generates mux_sel at the right time to send 32 bits of the 64 bits entry in the FIFO in the right sequence. The 32 bits of address pass through a parity generator block to generate the parity bit before being fed to the output buffer. If IAP (invert address parity) bit is set

by the software, then the address parity bit from the Parity Generator block is inverted to generate bad parity on the address bus. The state machine also generates S2U_RS to increment the read pointer in S2U_A FIFO.

The arbiter block maintains the following rules:

- The round robin bits keep the last master highest priority
- Any requestor can assert his request at any time
- If multiple requests were asserted, they arb using the current round robin state. At the end of the arb cycle, the round robin bits are set so the winner is the highest priority. For fairness, other requests should make the current master release his request (if asserted) within one or two transactions
- The system controller request always has the highest priority

When the system is woken from the SLEEP mode (power management mode), the SC asserts only the I_ARBRST_ signal to U2S (uparst_ is not asserted). This signal resets the arbiter logic and also enables a 6 bit counter. U2S is prevented from generating any requests for 64 upa clocks after the I_ARBRST_ signal is asserted so that as to give the processor enough time to come out of its reset.

8.4.1.2 UM Block Diagram

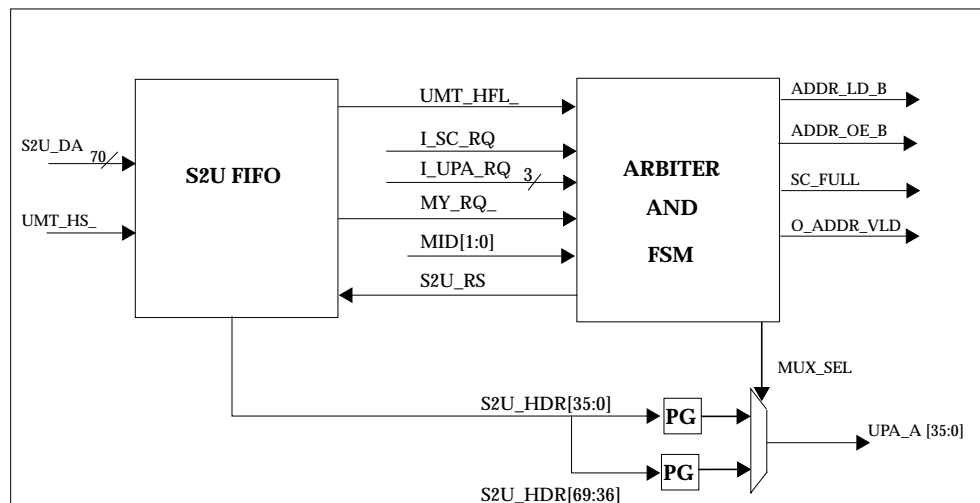


Figure 8-2 UM Block Diagram

8.4.2 UPA Slave (US) Description

8.4.2.1 US Overview

The UPA Master/Slave goes into the slave mode when it sees the Addr_valid signal asserted. The UPA_A bus is a 36 bit request bus including one Odd parity bit. All requests take two clock cycles to transfer on the UPA_A bus. The request is fed into the parity check block. All requests along with the parity result are stored in the FIFO. The USR_MRQ_ is asserted indicating to the PIO/DMA controller that there is a request in the U2S_A FIFO.

8.4.2.2 US Block Diagram

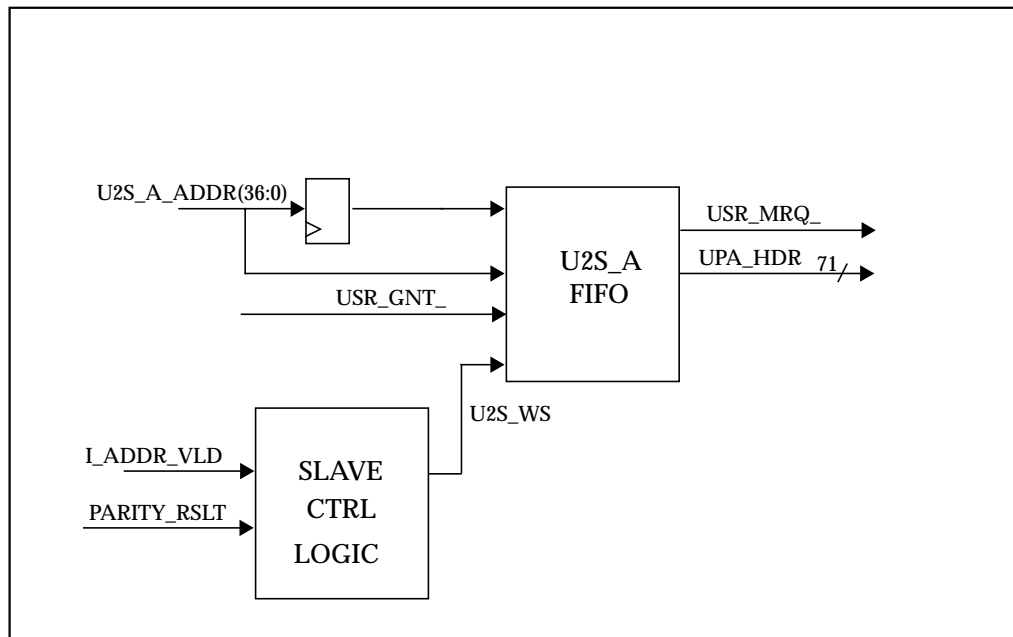


Figure 8-3 UPA Slave Block Diagram

8.5 PIO Logic Descriptions

The PIO block monitors diagnostic access into the UPA configuration registers.

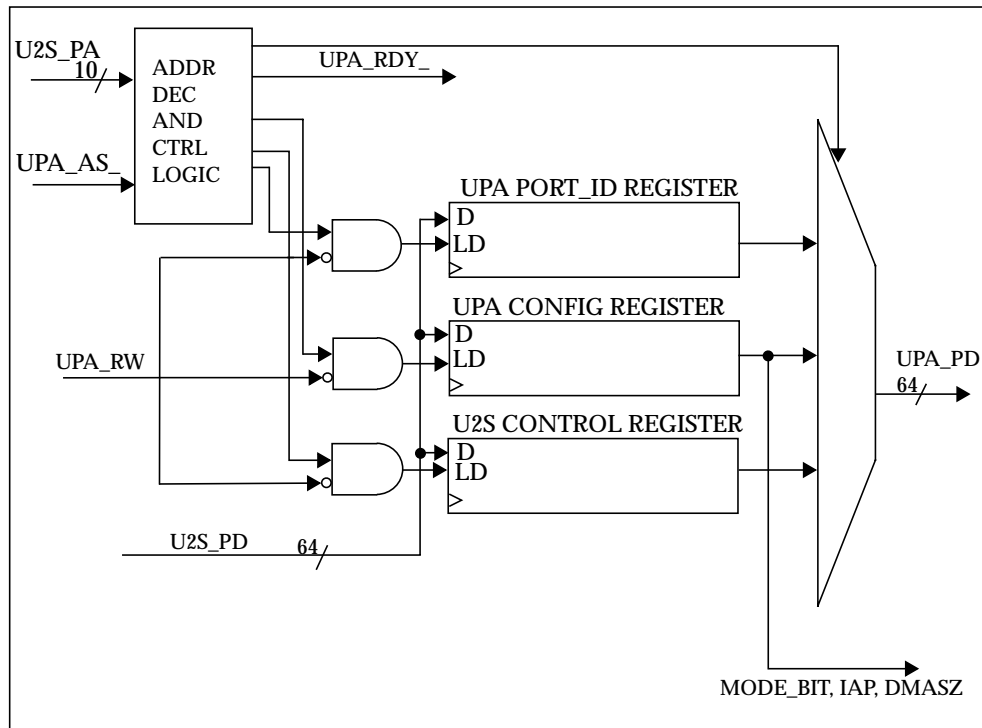


Figure 8-4 Block Diagram of PIO logic

9.1 Definition of Terms

Abbreviations, TLAs and ETLAs used in this section:

- U2S: The I/O Controller Chip
- SC: The System Controller ASIC
- DVMA: Direct Virtual Memory Access, (Sun's version of DMA)
- PIO: Programmed Input/Output
- UPA: Uniform Port Architecture
- S2U_D: SBus2 to UPA Address. Used for all DVMA cycles
- U2S_A: UPA Address for PIO cycles
- Line: 64 bytes, the cache line size
- Sub-line: Any access which is less than 64 bytes in size

9.2 UPA Reply Overview

The UPA Reply block is the U2S's interface to the UPA_Data bus. There are three functions that this block performs:

- Generates P_REPLY based on signals from the PIO unit
- Receives S_REPLY for both PIO and DVMA and manages data into and out of the UPA Data FIFOs
- Signals the appropriate PIO or DVMA control unit that data transfer has occurred

The UPA Reply unit manages replies to the SC chip. P_REPLY requests are received from the PIO Control Unit and are forwarded to the SC. For PIO reads, P_REPLY indicates to the SC that the U2S chip has read data ready in the PIO_RD FIFO. For PIO writes, P_REPLY indicates that the U2S has removed write data from the PIO_WR FIFO. P_REPLY is not used for DVMA.

S_REPLY is used for both PIO and DVMA cycles. For PIO reads, S_REPLY indicates that the SC is ready to read data out of the PIO_RD FIFO. For PIO writes, S_REPLY indicates that the SC is writing data into the PIO_WR FIFO. For DVMA reads, S_REPLY indicates that the SC is delivering data into the DVMA_RD FIFO, and for DVMA writes, S_REPLY indicates that the SC is ready to read data out of the DVMA_WR FIFO.

The UPA Reply unit has two clocks, the UPA Interface runs at the UPA clock speed, and the rest of the U2S runs at the U2S clock speed (which is twice the SBUS frequency). All the FIFOs are therefore written and read on two different clocks and signals going between the clock domains have to be synchronized. A full handshaking circuit is used to synchronize the signals going between the two clock domains.

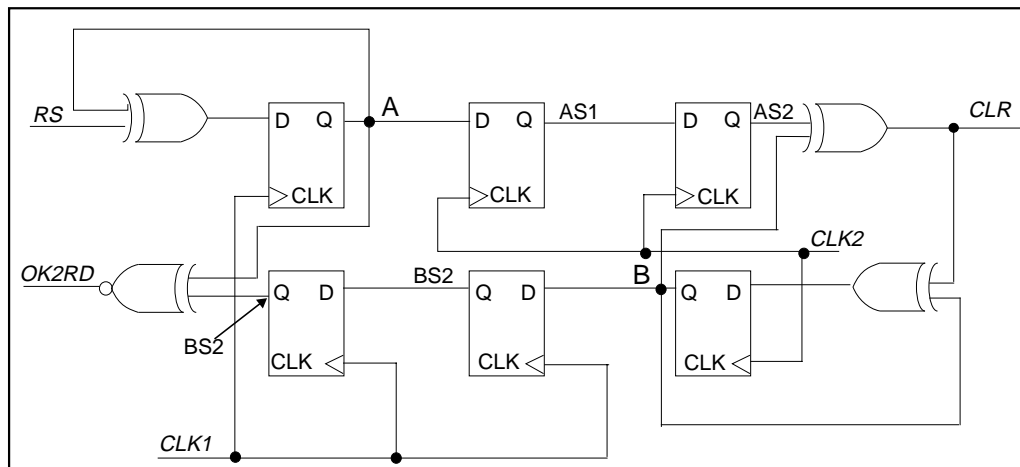


Figure 9-1 Full Handshaking Circuit

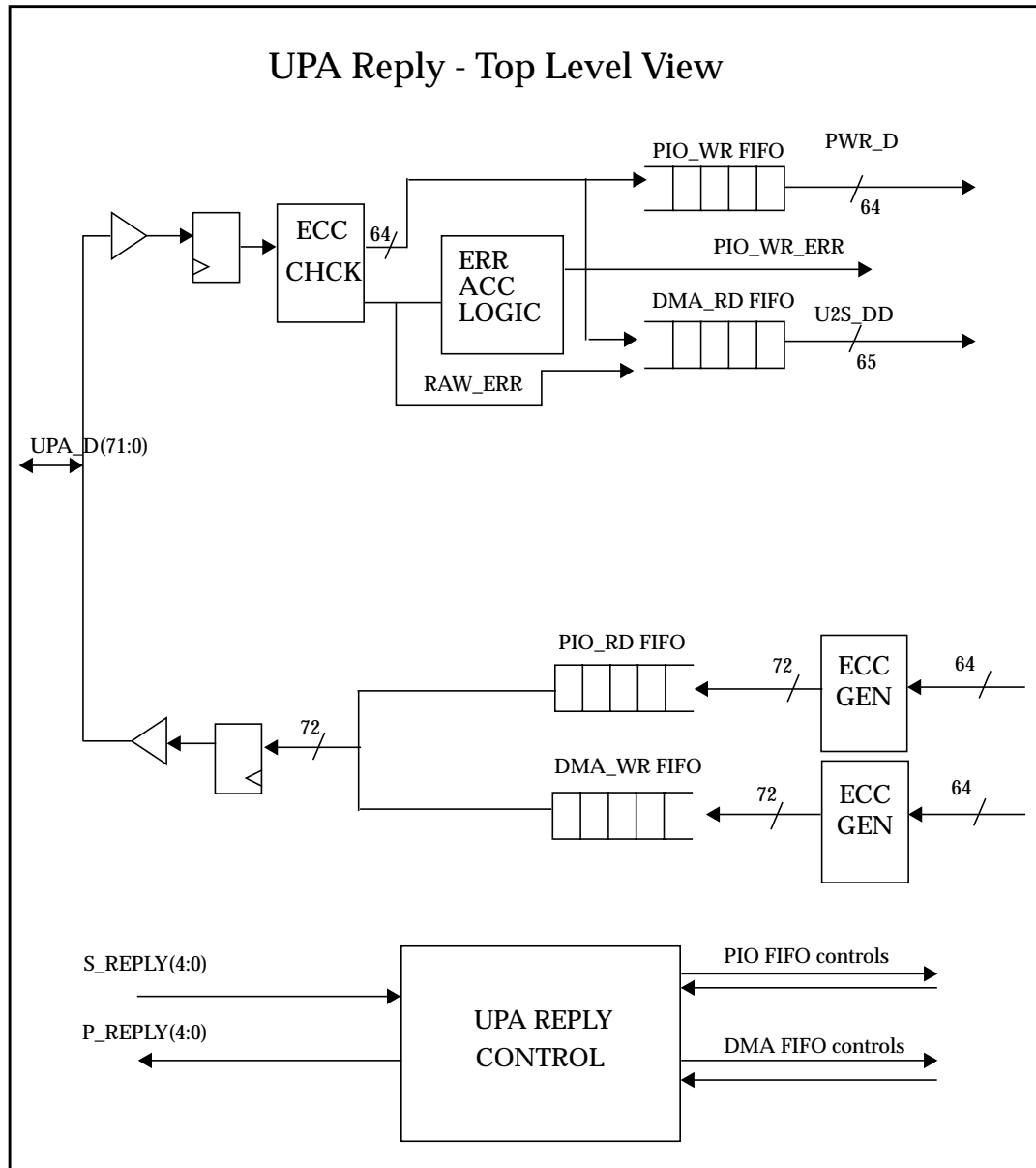


Figure 9-2 UPA Reply High Level Block Diagram

9.3 Signal Descriptions

Table 9-1 DMA Write Interface Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| UMT_WS_ | 1 | I | Write enable for the data FIFO |
| UMT_8B_ | 1 | I | Only 8 bytes are inserted in the data FIFO |
| UMT_32B_ | 1 | I | 32 bytes of data are valid in the data FIFO |
| UMT_DFL_ | 1 | O | Status of DMA_WR data buffer |
| S2U_DD | 72 | I | DMA Write Data including the check bits |

Table 9-2 DMA Read Interface signals

| Signal Name | Signals | I/O | Description |
|---------------|---------|-----|--|
| UMR_RS_ | 1 | I | Read enable for FIFO |
| UMR_RW | 1 | I | Indicates whether it is a read or write transaction |
| U2S_DD | 65 | O | DMA Read Data along with the error bit (UE) |
| DRD_TRAN | 1 | O | Indicates a DMA_RD transaction is progressing |
| PHADDR_DMA | 41 | O | Physical address of the DMA_RD transaction |
| UMR_SZ | 3 | I | Size of the DMA_RD transaction (this is received in the SIOCLK domain) |
| UMR_C | 1 | I | Indicates whether the DMA transaction was cacheable or not |
| DRD_SZ | 3 | O | Size of the DMA_RD transaction (this is sent to ECC unit in the UPACLK domain) |
| DMA_SRPLY | 1 | O | DMA related S_REPLY |
| DADX_VLD_PEND | 1 | O | This indicates when the address is valid for the DMA_RD UE |
| U2S_DA | 41 | I | Physical address of the DMA transaction |

9. UPA Reply Control

Table 9-3 PIO Write Interface Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| USR_WDV_ | 1 | O | Indicates that the data is present in the PIO_WR data FIFO |
| USR_RS_ | 1 | I | Read enable for the PIO_WR data FIFO |
| PWR_D | 64 | O | PIO Write Data |
| USR_GNT_ | 1 | I | Indicates that an entry has been read in the U2S FIFO |
| U2S_RS | 1 | O | Reader for the U2S FIFO |
| USR_RW | 1 | I | Indicates whether it is a read or write transaction |
| USR_WDE_ | 2 | O | Indicates that there is a an error (UE) in the data |
| PWR_TRAN | 1 | O | Indicates that a PIO_WR transaction is progressing |

Table 9-4 PIO Read Interface Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| UST_WS_ | 1 | I | Write enable for the data FIFO |
| UST_8B_ | 1 | I | Only 8 bytes have been inserted into the data FIFO |
| UST_DFL_ | 1 | O | Indicates that the PIO_RD data FIFO is full |
| S2U_PDG | 72 | I | PIO Read Data including the check bits |

Table 9-5 Miscellaneous

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| O_UPA_D | 72 | O | UPA Data output before the output register |
| UPADOE_B | 64 | O | Data Output Enable |
| ECCDOE_B | 8 | O | UPA ECC data output enable |
| UPADLD_D1B_ | 64 | O | UPA Data input register load enable |
| ECCDLD_D1B_ | 8 | O | ECC Data input register load enable |
| WRD_CNT | 3 | O | Keeps track of which word in a block is being checked |
| ECC_VLD | 1 | O | ECC valid for the incoming data |
| I_ECC_VLD | 1 | I | ECC valid for incoming data (valid only with S_REPLY) |
| DATA_VLD | 1 | O | Data valid to qualify UE or CE |
| I_UPA_DTST | 1 | I | UPA Data stall |
| UE | 1 | I | Raw non-correctable error from ECC unit |
| CE | 1 | I | Raw correctable error from ECC unit |
| ECC_CD | 64 | I | Corrected data from the ECC unit |
| FSR_RDY_ | 1 | O | Fault register has locked the error info |
| PADX_VLD_CE | 1 | O | PIO address valid for correctable error |
| PADX_VLD_UE | 1 | O | PIO address valid for non-correctable error |
| SIOCLK | 1 | I | U2S clock |
| UPACLK | 1 | I | UPA clock |
| SIORST_ | 1 | I | Reset with respect to U2S clock |
| UPARST_ | 1 | I | Reset with respect to UPA clock |

9. UPA Reply Control

Table 9-6 P_REPLY signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| O_P_REPLY | 5 | O | P_REPLY to the SC (before the output register) |
| UST_HS_ | 1 | I | Indicates that a P_REPLY for PIO has to be generated |
| UST_HFL_ | 1 | O | Cannot accept any P_REPLY |
| P_PHDR | 15 | I | Information to generate the right P_REPLY P_HDR[14:10] = MID P_HDR[9] = CLASS P_HDR[8:5] = TRAN_TYPE P_HDR[4] = INVALID SIZE P_HDR[3] = HDR_PERR P_HDR[2] = SBM_PERR P_HDR[1] = SBM_RTO P_HDR[0] = SBM_ERRV_ |

Table 9-7 S_REPLY signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|--|
| I_S_REPLY | 5 | I | S_REPLY from SC after the input register |
| S_REPLY_HDR | 2 | O | Output of S_REPLY FIFO S_REPLY_HDR[1] = ERR S_REPLY_HDR[0] = Data present or write ACK |
| UPA_RPL_ | 1 | O | Indicates that a S_Reply has been received |
| UPA_RPA_ | 1 | I | Asserted when an entry in the S_REPLY FIFO has been acknowledged |
| MODE_BIT | 1 | I | Indicates whether the UPA is running in a slow or fast mode |

9.4 UPA Reply Functional Description

The transaction set is composed of four components viz. P_REQ, S_REQ, P_REPLY and S_REPLY. P_REQ is generated by a UPA port and serviced by the System Controller (SC). The SC forwards the P_REQ to a slave UPA port if the address is in the address space of the slave UPA port.

S_REPLY and P_REPLY are acknowledgments, and constitute the reliable end-to-end transaction delivery mechanism in the interconnect. The replies are carried on unidirectional point-to-point wires between the SC and each UPA port.

The U2S header FIFO (described more in the UPA Master/Slave document) is only two deep. This FIFO stores the Requests received from the UPA Address bus. It can store two transactions.

The S2U header FIFO (described more in the UPA Master/Slave document) is also only two deep. There can be only one outstanding DMA_RD and one outstanding DMA_WR. It is limited to one read or write due to the availability of only one 64 byte buffer for read data and another single 64 byte buffer for write data. There is however an exception where two DMA_WR requests can be outstanding. This is when a 32 byte write request from SBUS is launched. In this case, the 32 byte request is broken into two 16 byte requests on the UPA.

9.4.1 Type of Requests, S_REPLY and P_REPLY

The following tables specify the legal request/reply combinations for U2S.

Table 9-8 U2S Requests - PIO Read Requests

| REQUEST | P_REPLY | S_REPLY |
|-------------|---------------------------------|---------|
| P_NCRD_REQ | P_RAS P_RERR P_FERR P_RTO | S_SRS |
| P_NCBRD_REQ | P_RAB P_RERR P_FERR P_RTO | S_SRB |

Table 9-9 U2S Requests- PIO Write requests

| REQUEST | S_REPLY | P_REPLY |
|-------------|---------|----------------|
| P_NCWR_REQ | S_SWS | P_WAS P_FERR |
| P_NCBWR_REQ | S_SWB | P_WAB P_FERR |

Table 9-10 S2U Requests - DMA Read Requests

| REQUEST | S_REPLY | P_REPLY |
|-------------|-----------------------|---------|
| P_RDD_REQ | S_RBS S_ERR S_RTO | none |
| P_RDO_REQ | S_RBU S_ERR S_RTO | none |
| P_NCRD_REQ | S_RAS S_ERR S_RTO | none |
| P_NCBRD_REQ | S_RBU S_ERR S_RTO | none |

Table 9-11 S2U Requests - DMA Write Requests

| REQUEST | S_REPLY | P_REPLY |
|-------------|----------------|---------|
| P_WRB_REQ | S_WAB | none |
| P_WRI_REQ | S_WAB | none |
| P_NCWR_REQ | S_WAS | none |
| P_NCBWR_REQ | S_WAB | none |
| P_INT_REQ | S_WAB S_INAK | none |

The UPA interface on the U2S could get two types of data from the UPA_Data bus. One could be the PIO_WR data and the other could be the DMA_RD reply data. The PIO_WR FSM and DMA_RD FSM handle the transfer of data from the UPA_Data bus to the respective FIFOs. The S_REPLY encoding determines which type of data is arriving. When the PIO controller unit has removed the data from the PIO_WR FIFO, a P_REPLY is sent to the SC indicating the same.

The U2S could send DMA_WR data or PIO_RD data on the UPA_Data bus. The SC sends an S_REPLY (in response to a P_REPLY -for PIO_RD- or DMA_WR request sent by the U2S) which signals to the respective state machines, DMA_WR or PIO_RD, to start driving the data. The state machines are described in more detail in Section 9.4.4, “DMA_WR FSM,” Section 9.4.5, “PIO_RD FSM Data transfer from the FIFO to the UPA_D bus,” and Section 9.4.8, “PIO_RD and DMA_WR Data transfer from the respective controller units to the FIFO.”

9.4.2 PIO_WR FSM

This state machine handles the transfer of data from the UPA Data Bus to the PIO_WR data FIFO associated with PIO_WR requests. SC issues S_SWS or S_SWB reply to U2S data from the UPA_Data bus. The S_REPLY from the SC is decoded to determine whether it is a 16 byte write (S_SWS) or 64 byte write (S_SWB). The data first passes through the ECC checking unit before it enters the FIFO. The ECC result is logged only after all the data from that transaction has been checked.

9.4.2.1 PIO_WR FSM Block Diagram

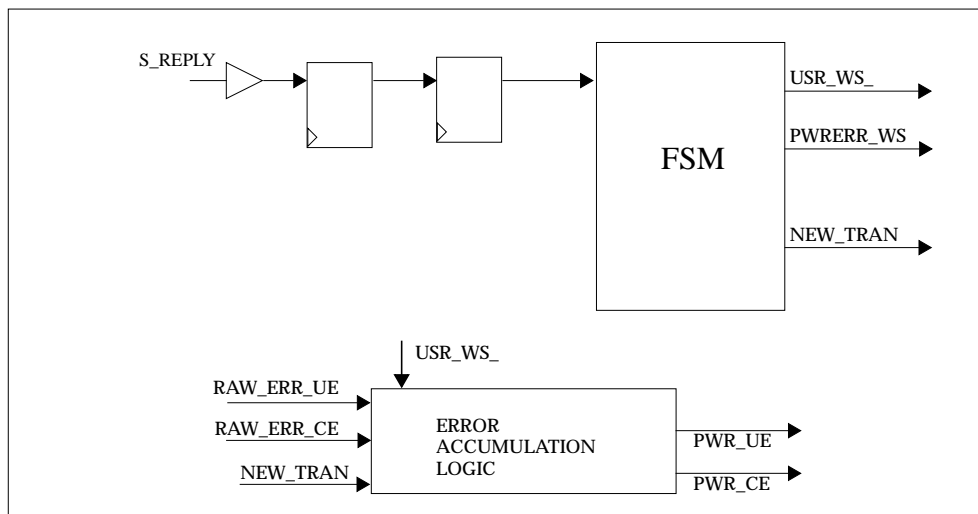


Figure 9-3 FSM Block Diagram

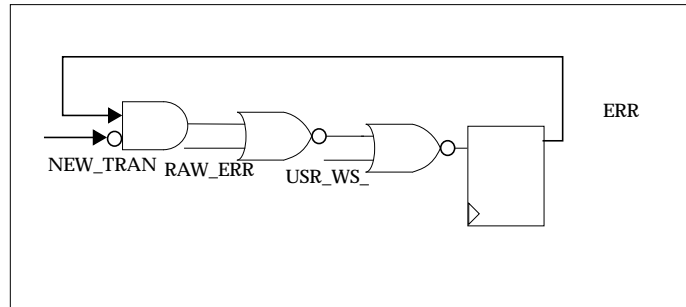


Figure 9-4 Error Accumulation Logic

9.4.3 DMA_RD FSM

DMA_RD requests are sent by U2S to the SC through the S2U_A FIFO. The SC then provides the data with a S_RAS or S_RBU or S_RBS reply. The DMA_RD FSM handles the transfer of data associated with DMA_RD replies. When this state machine detects the relevant S_REPLY (S_RAS or S_RBU or S_RBS) from the SC, the S_REPLY is further decoded to determine whether it is a 16 byte or 64 byte transfer. The data first passes through the ECC checking unit before it enters the FIFO. The ECC result is logged along with the data in the FIFO.

9.4.4 DMA_WR FSM

This state machine handles the transfer of data from the DVMA_WR data FIFO to the UPA_Data bus. When the SC receives a DMA_WR request from the U2S, it sends S_WAB (16 byte write) or S_WAS (64 byte write) to indicate that the U2S can start driving the data. When only 8 bytes of data are inserted into the DMA_WR data FIFO, UMT_8B_ is asserted, which indicates the state machine to duplicate the same 8 bytes in the second cycle of data transfer. When 32 bytes of data are inserted (for a 32 byte DMA Non cached Write request from SBUS), the corresponding request is split as two 16 byte NCWR requests, and the data is drained accordingly for the two S_REPLYs received. If the DMA_WR request was an interrupt request, the SC could reply with S_INAK (indicating that there is no room in the destination UPA port to accept the mondo vector). The S_REPLY is therefore decoded to transition to the proper state (WRS or WRB or ABRT). When the S_REPLY decodes to a S_INAK, the FIFO has to be drained but the output buffers are not driven.

9.4.5 PIO_RD FSM Data transfer from the FIFO to the UPA_D bus

This state machine handles the transfer of data from the PIO_RD data FIFO to the UPA Data bus. When SC sends S_SRS or S_SRB, it signals the U2S to drive PIO_RD data on the UPA_Data bus. The S_REPLY is decoded to determine whether it is a 16 byte write or 64 byte write. When only 8 bytes of data are inserted into the PIO_RD data FIFO, UST_8B_ is asserted which directs the state machine to duplicate the 8 bytes in the second cycle of data transfer.

9.4.6 PIO_WR Data transfer from the FIFO to the PIO controller unit

The PIO controller unit gets a data valid signal (this is sent when all the data has been checked through the ECC unit). If it was a 64 byte PIO_WR for example, data valid is asserted only after all 64 bytes have been checked for ECC errors. The USR_RS_ would advance the head pointer in the PIO_WR data FIFO.

9.4.7 DMA_RD Data transfer from the FIFO to the DMA controller unit

The S_REPLY is decoded and if it decodes to a DMA type S_REPLY it is inserted into the S_REPLY FIFO. The DMA controller unit then reads the data from the DMA_RD FIFO. The UMR_RS_ advances the head pointer in the DMA_RD data FIFO.

9.4.8 PIO_RD and DMA_WR Data transfer from the respective controller units to the FIFO

The PIO or DMA controller unit will transfer data to the PIO_RD or DVMA_WR data FIFO respectively. The PIO_RD data FIFO is written into when UST_WS_ is asserted. The DMA_WR data FIFO is written into when UMT_WS_ is asserted. Along with the data arrival the PIO controller unit sends information about the packet to generate the P_REPLY for PIO_RD.

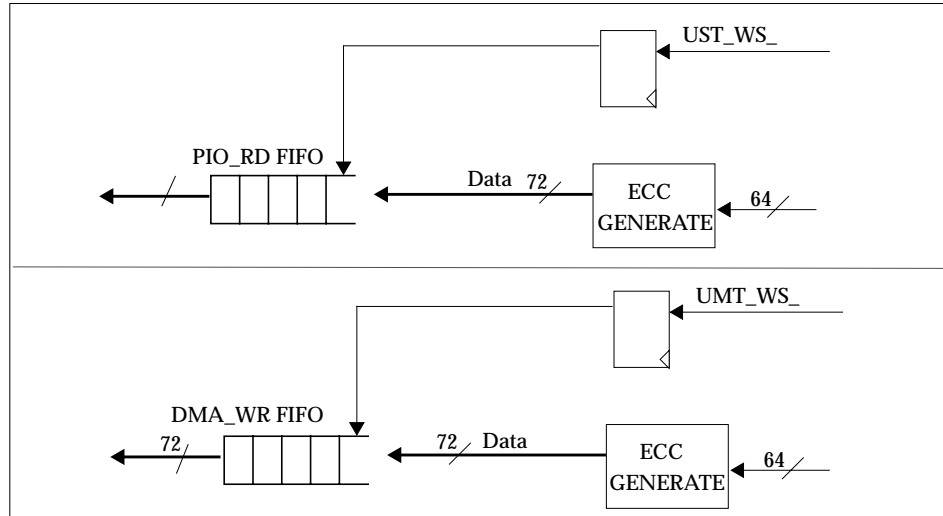


Figure 9-5 Data path flow from controller units to FIFO

9.4.9 P_REPLY

The P_Reply signals to the System Controller the status of previous requests. In a two cycle P_REPLY packet, the first cycle contains the P_REPLY type and the class bit. The second cycle contains the MID of the requesting UPA master port to which this is a reply for.

In a single cycle P_REPLY, the second cycle is dropped.

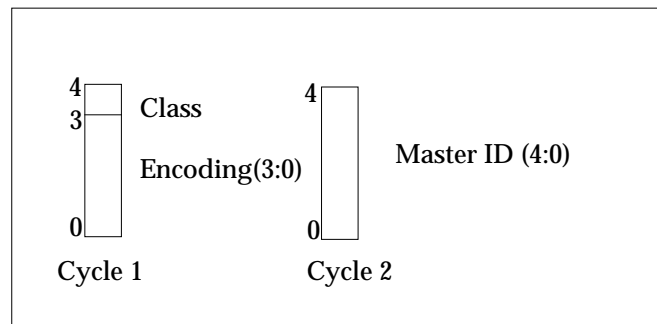


Figure 9-6 P_REPLY FORMAT

The information needed to generate the right P_REPLY is sent by the bus controller unit on P_PHDR. This is decoded to format the right P_REPLY.

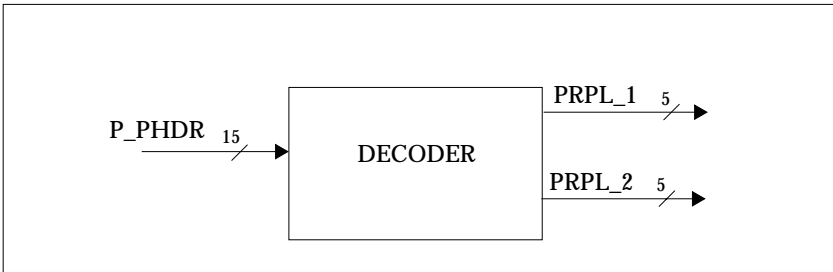


Figure 9-7 P_REPLY Decoder

9.4.9.1 P_REPLY FSM Block Diagram

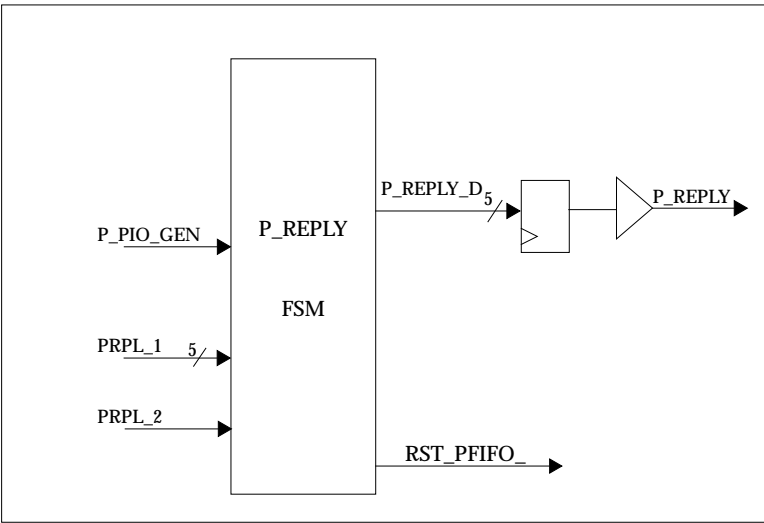


Figure 9-8 FSM Block Diagram

9.4.10 S_REPLY

The S_REPLY is the acknowledgment generated by SC to a UPA port. The SC observes strong ordering within each class and delivers the S_REPLY in the same order as the requests sent to it.

The S_REPLY for DMA transactions are queued so that the DMA controller unit can take the proper action to clear its scoreboard. A two bit FIFO maintains the result of the S_REPLY. When an S_REPLY is received from SC, it is decoded to determine whether it should be put in the FIFO since only DMA type replies need be seen by the DMA controller unit.

S_REPLY for DMA_RDs could be S_RERR | S_RTO (error) or S_RBU | S_RAS (data is arriving). S_REPLY_HDR[1] logs if there is an error or not. S_REPLY_HDR[0] indicates whether there is data or not. S_REPLY_D is the data bus into the FIFO.

S_REPLY for DMA_WRs could be S_WAS | S_WAB (write acknowledge) or S_INAK (mondo vector Nack).

S_REPLY_H[0] logs if the write was an ACK or Nack. In writes S_REPLY_H[1] bit has no role.

On DMA_RD Replies, the arrival of an S_REPLY indicates that data will arrive in the following cycles. Data is written into the FIFO in UPA clock and read out in the U2S clock. If the U2S clock is substantially faster than the UPA clock, the DMA controller could start reading the FIFO before all data has arrived there. To solve this problem, there is a programmable bit called the MODE bit in the U2S control register (see Chapter 4, “Programming Model”), which indicates whether the UPA is running in the fast or slow mode. In the slow mode, the S_REPLY is written into the FIFO delayed by ten UPA clocks (if it is a 64 byte data transfer accounting for two cycles of data stall) to ensure that all the data has arrived in the DMA_RD FIFO before the DMA controller starts reading the data. There is an additional restriction that when UPA is running slower than the U2S clock, we can have only one outstanding DMA request.

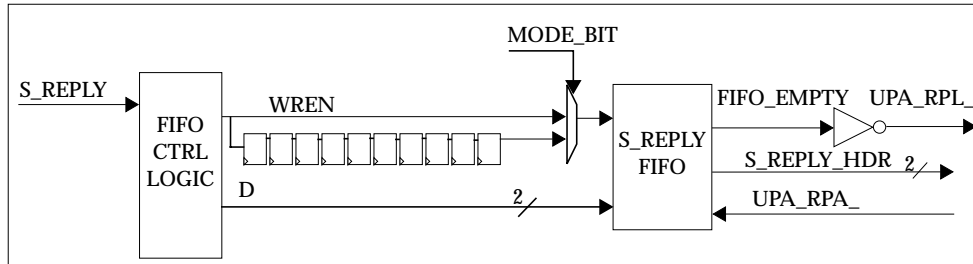


Figure 9-9 S_REPLY Block Diagram

10.1 Definition of Terms

- CE: Correctable Error
- UE: Uncorrectable Error
- ECC: Error Correction Code

10.2 Overview

All UPA data transfers (except certain ports like the graphics port) in the UltraSPARC system are ECC protected. ECC generation is always enabled. ECC checking can be disabled through the ECC Control Register. The ECC is generated by the Master on writes and checked by the Master on reads. The slave stores ECC with the data (for example: if the slave is a memory) or must generate ECC when read.

- ECC hardware is common among all UPA ports. However, the error handling mechanism will be different between processor and IO
- All UPA data transfers (except when the Graphics port delivers data) carry ECC. It is either obtained from storage (for example: DRAM), or generated by the interface providing the data (Processor, UDB chips, U2S)
- ECC is done on a 64-bit data boundary with 8 Check Bits
- ECC implements SEC/DED/S4ED error correction code. This code corrects any single-bit error, detects any two-bit errors, and detects three or four bit errors within a nibble
- Memory is always cacheable so partial writes to memory are not allowed
- When a correctable error is detected, the corrected data is delivered to the requesting master. Software must update the main memory with the corrected data if the data source is from the memory
- Memory scrubbing will be done through software. No hardware scrub is supported

10.2.1 ECC Overview

The major blocks of ECC are:

- Syndrome Generator for SEC/DED/S4ED code
- Error Detection
- Error Decode and Correction
- Configuration Registers and Mondo Vector Generation

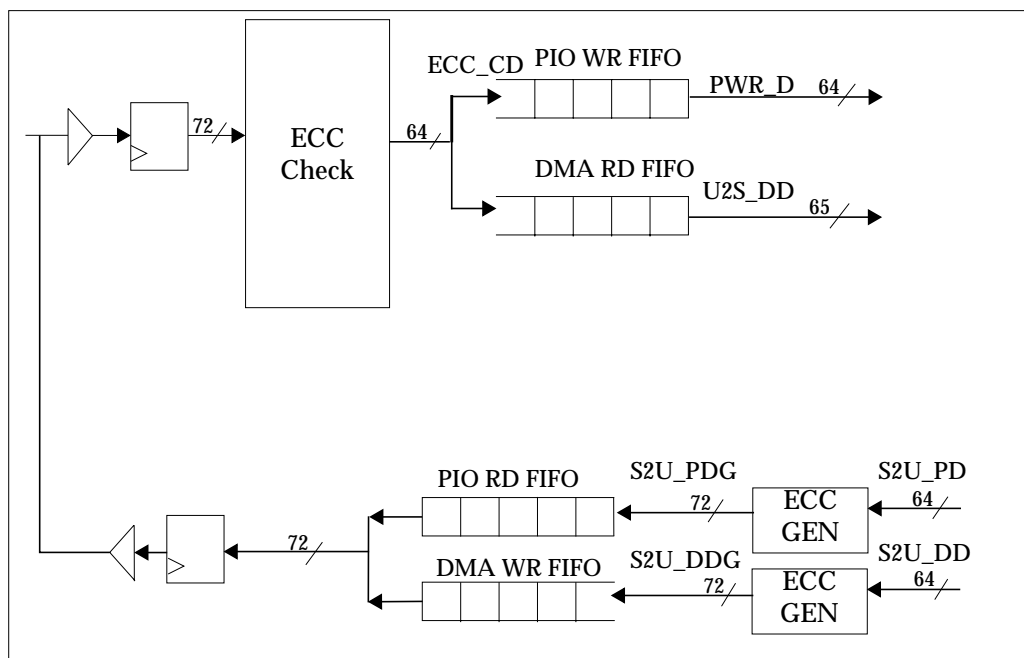


Figure 10-1 Interface Block Diagram of ECC Unit

10.2.2 Overview Block Diagram of the ECC Unit

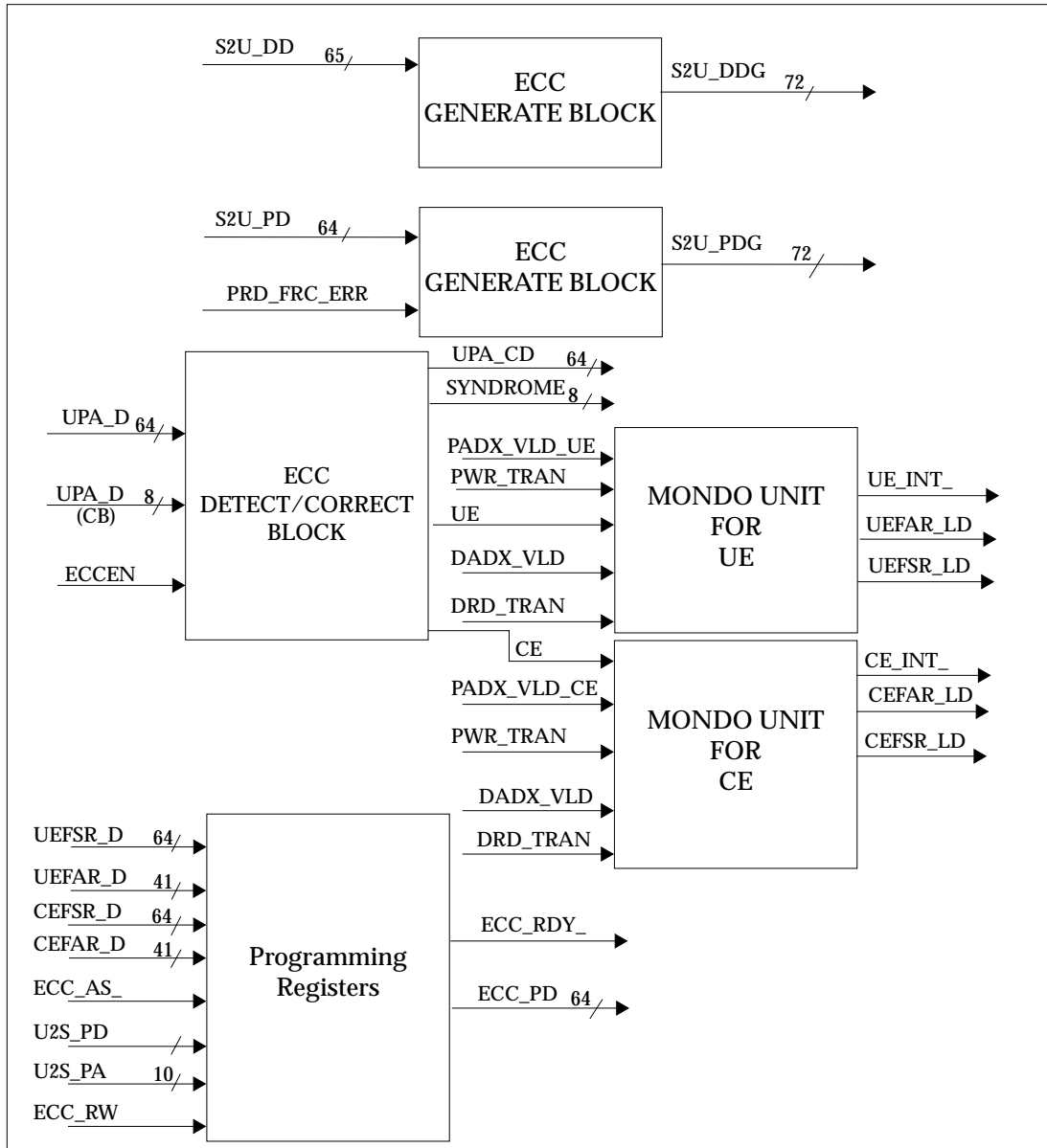


Figure 10-2 Block Diagram of ECC Unit

10.3 Signal Descriptions

Table 10-1 ECC Detect/Correct Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| UPACLK | 1 | I | UPA clock (83 Mhz) |
| SIOCLK | 1 | I | U2S clock (50Mhz) |
| SIORST_ | 1 | I | U2S clock reset |
| UPARST_ | 1 | I | UPA clock reset |
| ECC_RW | 1 | I | Read/Write control for the ECC configuration registers |
| U2S_PA | 41 | I | Specifies address to read or write or the PIO fault address |
| ECC_AS_ | 1 | I | Address strobe to indicate when U2S_PA is valid |
| ECC_RDY_ | 1 | O | When asserted, indicates that ECC has driven valid data in response to PIO read operation |
| U2S_PD | 64 | I | PIO write data |
| ECC_PD | 64 | O | PIO read data |
| UMT_WS_ | 1 | I | DMA Write Data valid |
| UST_WS_ | 1 | I | PIO Read Data valid |
| PHADDR_DMA | 41 | I | 41 bit DMA address to be sent as part of the Mondo vector packet |
| UE_INT_ | 1 | O | To signal Mondo dispatch arbiter that ECC has a mondo vector related to UE to send |
| CE_INT_ | 1 | O | To signal Mondo dispatch arbiter that ECC has a mondo vector related to CE to send |
| UE_VLD | 1 | O | Indicates an uncorrectable error when asserted |
| CE_VLD | 1 | O | Same as UE_VLD but indicates correctable error |
| I_UPA_D | 72 | I | Data bus from UPA after the Input Register |
| UPA_CD | 64 | O | Corrected data from the ECC check unit |
| PWR_TRAN | 1 | I | PIO Write Transaction taking place |
| DRD_TRAN | 1 | I | DMA Read Transaction taking place |
| S2U_DD | 65 | I | Incoming DMA_WR data |

Table 10-1 ECC Detect/Correct Signals

| Signal Name | Signals | I/O | Description |
|-------------|---------|-----|---|
| S2U_PD | 64 | I | Incoming PIO_RD data |
| PRD_FRC_ERR | 1 | I | Force bad ECC on PIO read data |
| S2U_PDG | 72 | O | Outgoing PIO_RD data with check bits |
| S2U_DDG | 72 | O | Outgoing DMA_WR data with check bits |
| PADX_VLD_UE | 1 | I | PIO address valid for UE |
| PADX_VLD_CE | 1 | I | PIO address valid for CE |
| DADX_VLD | 1 | I | DMA address valid |
| ECC_VLD | 1 | I | Indicates whether the data has valid ECC |
| DATA_VLD | 1 | I | Data valid to qualify UE and CE |
| PIO_MID | 5 | I | MID of the PIO Write transaction |
| MID | 5 | I | U2S MID (for recording DMA_RD error) transaction |
| USR_SZ | 3 | I | Size of transaction for PIO Write |
| DRD_SZ | 3 | I | Size of transaction for DMA Read |
| RTO_RRP_ | 1 | I | Indicates that it is a partial DMA write |
| WRD_CNT | 3 | I | Indicates which line of the 16 bytes or 64 bytes is being checked |

10.3.1 ECC Code

UltraSPARC micro-processors implement a SEC/DED/S4ED error correction code based on a paper by Kaneda (IEEE Trans. on Computers, Aug 84). This code provides correction of any single-bit error on a 64-bit data and 8 check bits. In addition, it provides detection of any two-bit errors, as well as three or four bit errors within a nibble.

Table 10-2 SEC/DED/S4ED Syndrome Encoding

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

The 8 bit SYNDROME[7:0] field is interpreted as follows:

* - no error

Number - the bit number of single error (refers to the 64-bit data)

C-number - the bit number of single error (refers to the 8 check bits)

D - double-bit error

T - triple-bit error

Q - quadruple-bit error

M - more than 4 errors

10.4 ECC Functional Description

10.4.1 ECC Check Description

10.4.1.1 Overview

ECC Check circuit is used to check correctness of UPA data. Specifically for U2S, it is active only during PIO Write and DVMA Read operations.

Correction can be disabled through a configuration register whereas ECC generation is always turned on. Any single-bit error can be detected and corrected. Any double-bit errors can be detected. In addition, three-bit or four-bit errors can be detected as well, provided all the bits reside in the same nibble.

Upon receiving data from the UPA, the ECC Check circuit starts by calculating its 8-bit syndrome. If any one of the 8-syndrome bits happens to be a “1”, then either the incoming data or the check-bit is incorrect. The actual number and the position of the errors are encoded according to the Syndrome Encoding table provided above. The Correctable Error (CE) flag will be set, if there is exactly one error in the data. On the other hand, the Uncorrectable Error (UE) will be set if there is more than one error.

If correction is enabled, the syndrome bits generated above will be decoded to determine which bit from the entire 72-bit entity is wrong, if there is any. The data will be corrected if it has exactly one bit error. Multiple bits error, when detectable using the SEC/DED/S4ED code, will simply be reported.

During PIO Write and a DVMA RD, both CE and UE will trigger U2S to send an interrupt to the processor. The fault address, syndrome bits and other information relevant to the error will be stored in FSR and FAR registers. The processor can then do a PIO read on these registers.

10.4.1.2 ECC Check Block Diagram

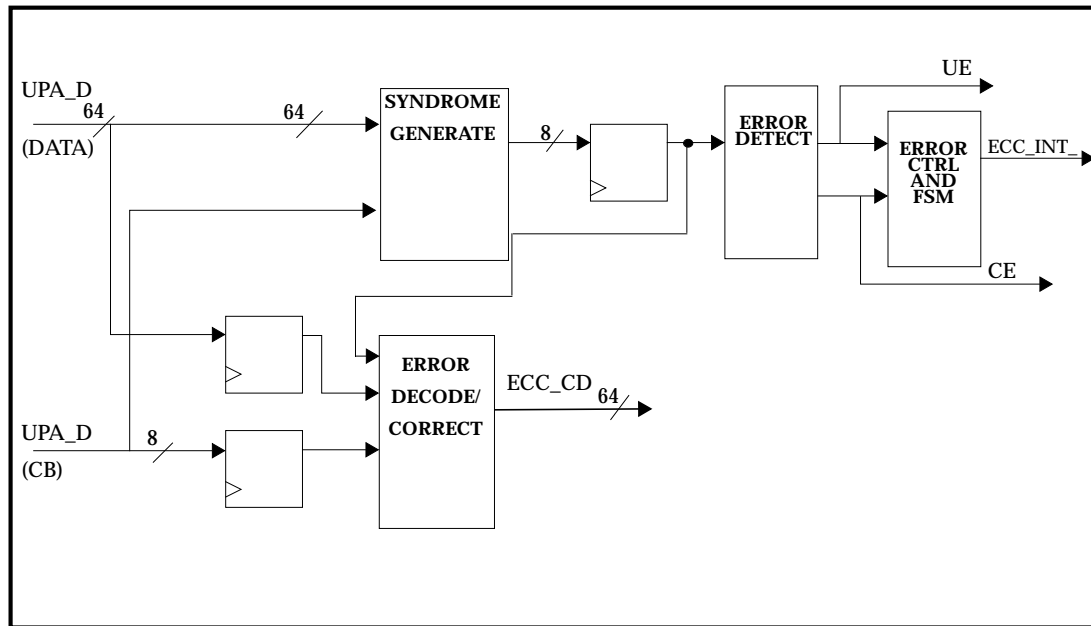


Figure 10-3 Check Block Diagram

10.4.2 ECC General Description

10.4.2.1 Overview

ECC Generate circuit is used to generate syndrome/check bits for the UPA data. Specifically for U2S it is active only during PIO Read and DVMA Write operations. 8 bits of syndrome is generated per 64-bit data boundary and 8 check bits. Since U2S does not maintain ECC in its internal datapath, the 8 redundant bits for the ECC will be forced to zeros in order to generate the syndrome bits.

Check bits generation in U2S is always turned on whenever there is a PIO Read or DVMA Write. All data originating from the SBus is CRC-checked by the SBus logic. As such, the ECC Generate circuit always assumes error-free data whenever it is instructed to generate ECC. In other words, the only function of ECC Generate circuit is to generate check bits; it doesn't have anything to do with error checking. There are certain exceptions however when we force bad ECC on the outgoing data.

The first case is as follows: The U2S issues a Read to Own transaction on the UPA in order to do a partial write. The data received from memory has a bad ECC on one of the eight bytes. The data which needs to be merged does not overwrite the eight byte data which has the bad ECC. In this case we would force bad ECC on the data which originally was corrupt. However if we happen to overwrite the data which happened to have bad ECC, then correct ECC is generated.

The second case is as follows: If a parity error is detected on PIO read data, we force bad ECC on the corresponding data when sending out to the UPA.

10.4.2.2 ECC Generate Block Diagram

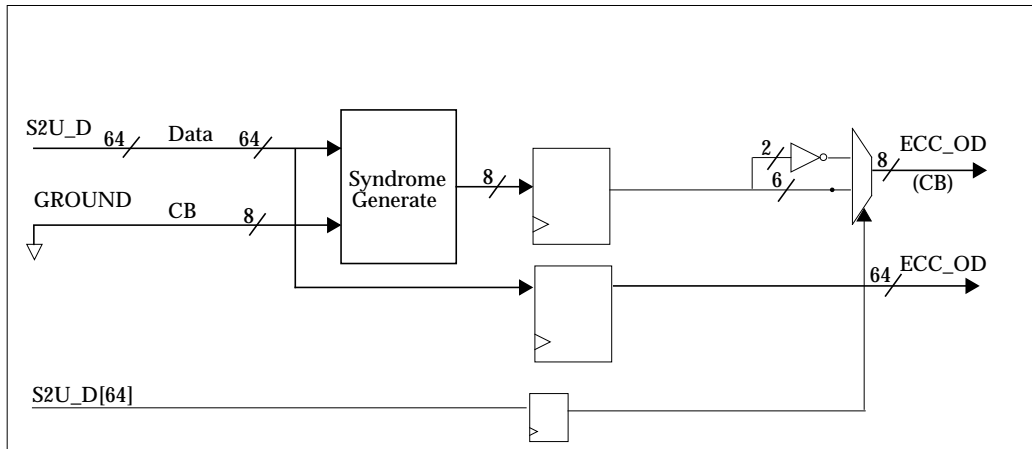


Figure 10-4 ECC Generate Block Diagram

10.5 ECC Mondo Vector Unit Description

10.5.1 Overview

The main function of this block is to request and send interrupts in the event of a correctable or uncorrectable error on PIO Writes and DVMA Reads. The ECC has two Mondo units, one to handle UE and one for CE. The main function of this unit is to log the address and status of the error and to send an interrupt to the Mondo Dispatch unit. The FSM described in Section 10.5.2, "FSM Block Diagram," is instantiated twice, one to handle UE and another to handle CE.

10.5.2 FSM Block Diagram

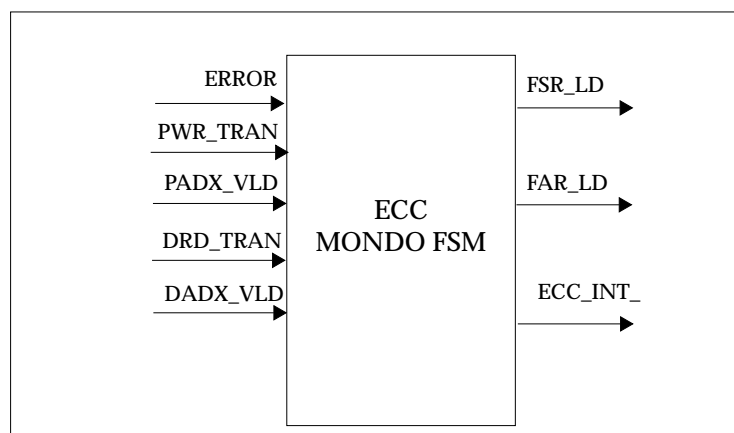


Figure 10-5 FSM Block Diagram

10.5.3 State Diagram

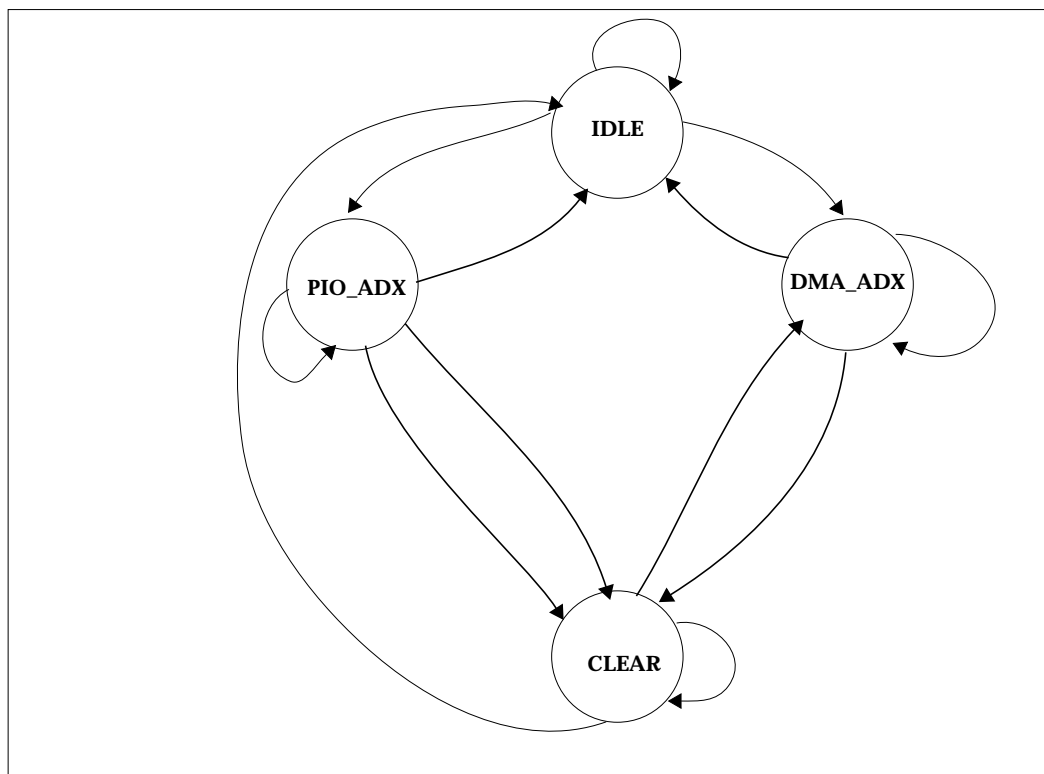


Figure 10-6 MV_RQ State Diagram

10.6 PIO Logic Descriptions

The PIO block monitors diagnostics access into the ECC configuration registers.

10.6.1 Block Diagram of PIO Logic

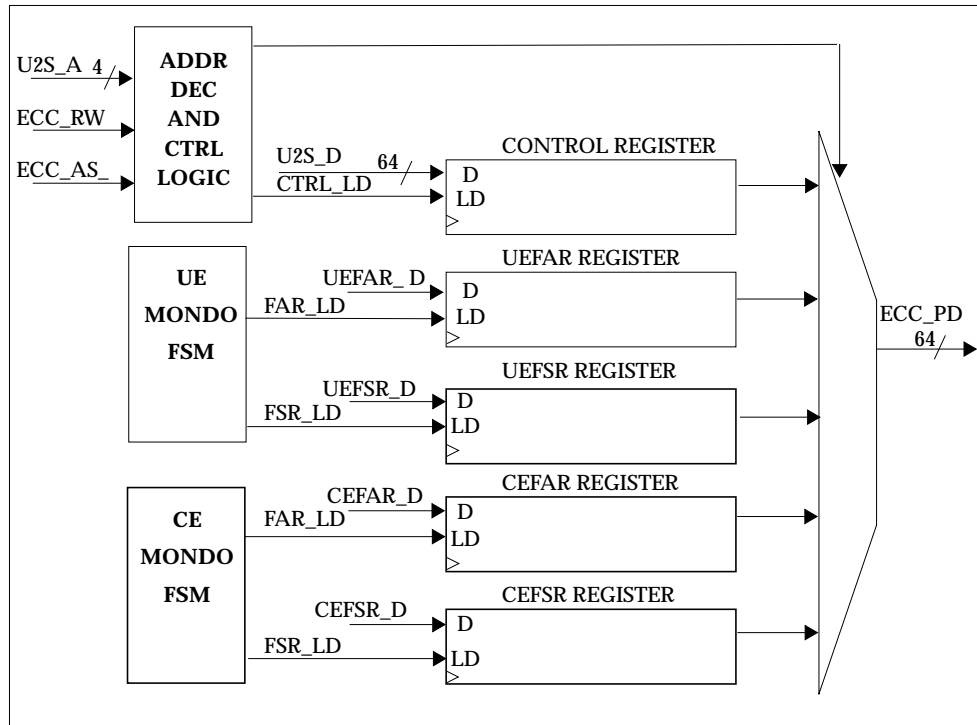


Figure 10-7 Block Diagram of PIO logic

11.1 Definition of Terms

Abbreviations, TLAs and ETLAs used in this section:

- DCC: Merge Buffer (formerly known as DMA Cache)
- RMW: Read-Modify-Write
- SC: System Controller ASIC
- DVMA: Direct Virtual Memory Access (Sun's version of DMA)
- PIO: Programmed Input/Output
- UPA: Universal Port Architecture
- Line: 64 bytes, the cache line-size
- Sub-line: Any access which is less than 64 bytes in size
- WL: A DMA Write of 64 Bytes (Write Line)
- WS: A DMA Write of less than 64 Bytes (Write Sub-line)
- RD: A DMA Read of any size
- RTO: UPA Read-to-Own
- RTD: UPA Read-to-Discard
- WB: A cache initiated Writeback on UPA
- WI: UPA Write-Invalidate
- CB: Generic Copyback Request from the SC
- CBI: Copyback-Invalidate from the SC
- CBS: Copyback-to-Share from the SC
- CBD: Copyback-to-Discard from the SC
- INV: Invalidate Request from the SC
- DVMA is synonymous with DMA

Prefixes to Signal Names

- U2S - UPA to SBus buses (address and data)
- S2U - SBus to UPA buses (address and data)
- DCC - Merge Buffer (was DMA Cache)
- MDU - Mondo Dispatch Unit
- MMU - IOMMU
- SBM - SBus Module
- STC - Streaming Cache

Example of Signal Names

- U2S_PA - UPA to SBus PIO Address bus
- U2S_PD - UPA to SBus PIO Data bus
- U2S_DA - UPA to SBus DMA Address bus
- U2S_DD - UPA to SBus DMA Data bus

11.2 DMA Merge Buffer Overview

Main memory in the UltraSPARC architecture is only accessible in cached 64-byte quantities. DMA devices (particularly older ones on the SBus) often need to access less than 64 bytes at a time. For DMA reads, this is not a problem; the U2S reads the entire 64-byte line and discards what is not needed. However for DMA writes the U2S must first read the line, then, merge in the new data and finally write the result to memory.

To reduce complexity and save chip real estate and verification time, the Merge Buffer will write the merged data immediately back to memory upon receiving the 64 byte line from the UPA. The SC will block all subsequent accesses to that line until the U2S has completed the Writeback. This allows the U2S to perform partial writes to cacheable memory without having to participate in the system coherency scheme. Since the SC blocks all requests to that line, the U2S will never receive Copybacks and Invalidates for that line.

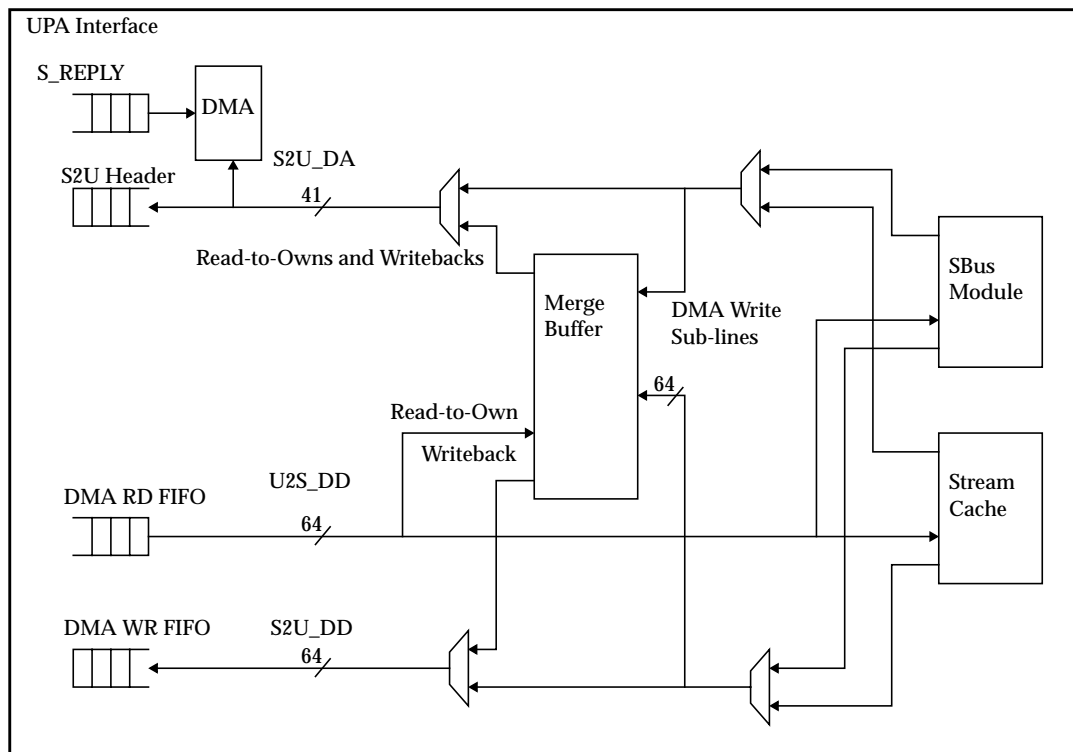


Figure 11-1 Merge Buffer Overview

11.3 Functional Description

The Merge Buffer's role is to service sub-line (less than 64 Bytes) writes. The basic operations the DCC must perform are listed below:

- Merge data from sub-line writes into a 64 Byte cache line
- Write the line back to Memory

This section is divided into four main parts:

- Section 11.3.1, "Cache Coherency Protocol," gives a brief overview of the cache coherency protocol used in the system and how the U2S uses a subset of this protocol
- Section 11.3.2, "Operation," gives an explanation of the Merge Buffer's operation
- Section 11.3.3, "Servicing DMA Partial Write Requests," describes how the Merge Buffer handles DMA Requests

11.3.1 Cache Coherency Protocol

This section gives a brief overview of the Ultra SPARC Cache Coherence Protocol. For a more detailed description, please refer to the UPA Interconnect Architecture Document.

11.3.1.1 System Cache Line States

The UltraSPARC caches use a 5-state coherence protocol known as MOESI, where the five possible etag (cache tag) states are:

- Invalid (I)
- Shared Clean (S)
- Exclusive Clean (E)
- Shared Modified (O)
- Exclusive Modified (M)

For the purpose of maintaining cache consistency via snooping, the SC chip maintains the duplicate tags (dtags) using a 4-state coherence protocol known as MOSI, where the four possible states are:

- Invalid (I)
- Shared Clean (S)
- Shared Modified (O)
- Exclusive & Potentially Modified (M)

Notice that in the dtag protocol, the Exclusive Clean and Exclusive Modified states have been merged. We are assuming that when you read a line you will probably modify it in the future. Modifying the line will not require any additional coherence transactions.

11.3.1.2 The Merge Buffer Line States

The Merge Buffer does not participate in system coherency. Although it may temporarily own a line in the M state it will never keep the line or service coherent transactions for that line. See Section 11.3.2, “Operation,” for a further explanation.

11.3.2 Operation

The Merge Buffer services only one type of request: DMA Writes of less than 64 bytes (WS) from the SBus and Streaming Cache. All other DMA activity bypasses the Merge Buffer. It also initiates writeback requests to the UPA.

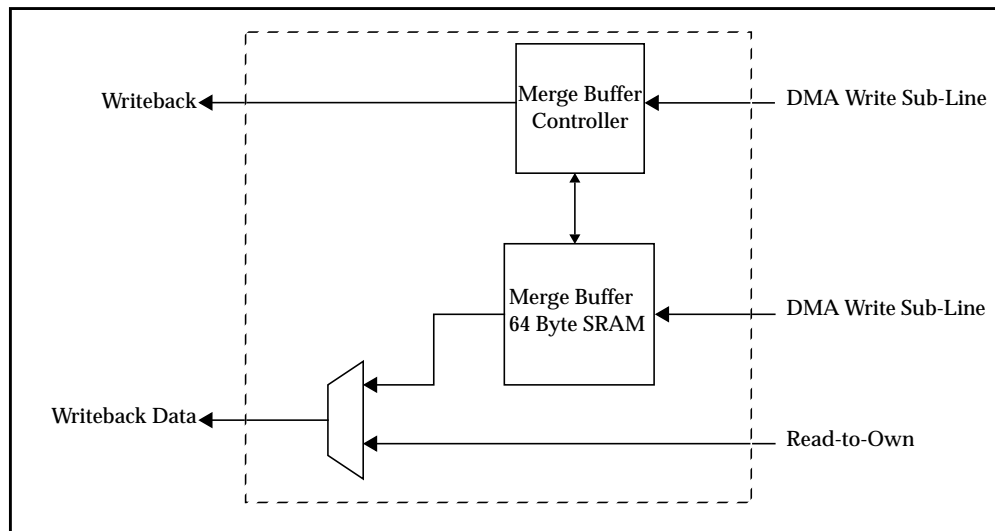


Figure 11-2 Merge Buffer Functional Block Diagram

The Merge Buffer accepts partial write data from the SBus and Streaming Cache and places the data into the Merge Buffer SRAM. At the same time, a Read-to-Own for the line is issued onto the UPA.

When the Read-to-Own data comes back from the UPA, the Merge Buffer issues a Writeback. At the same time, it takes the Read-to-Own data from the UPA DMA Read Fifo and merges it with the partial write data in its SRAM. The merged block is sent as the Writeback data to the UPA DMA Write Fifo. During this time (when the Read-to-Own data returns and the Writeback is waiting to be completed), the U2S technically owns the cache line (in the M state). However, the System Controller makes note of this line in its transaction score-board (see SC Manual), and blocks any subsequent transaction to that line until the Writeback completes. Thus U2S doesn't receive any coherent transaction for that line, and does not participate in system coherence. After the writeback completes, all transactions blocked to that line then proceed. Again, since U2S no longer owns the line, it will not see any coherent traffic to that line.

The general idea here is to simplify the complexity in U2S by essentially "freezing" the line until U2S writes the data back to memory. The U2S strategy is to write the data back as fast as possible, so that the time in which the line is "locked" is minimal. Also, by not having to participate in system coherence, the SC does not have to keep duplicate tags for the U2S.

Also note that during the time that the U2S issues the Read-to-Own and the time that U2S issues the Writeback, the U2S will not issue any other master transactions including interrupts. All DMA traffic to UPA is blocked until the Read-to-Own/Writeback combination is completed.

11.3.3 Servicing DMA Partial Write Requests

11.3.3.1 Merge Buffer Policies

1. All and only WSs go through the Merge Buffer.
2. Ordering (SSO) is preserved within the U2S. To insure SSO, when the Merge Buffer receives a WS, all DMA activity from any source is stalled until the Writeback of the line has completed. All DMA activity includes all DMA Reads and Writes from SBus and the Streaming Cache, Interrupts from the Mondo Dispatch Unit, and Table Walks from the IOMMU.

11.3.3.2 Partial Write Transaction

This section describes the servicing of a partial write by the Merge Buffer, (see the timing diagram, Figure 11-3, “Servicing a Partial Write”). Each cycle in the illustration is explained herein.

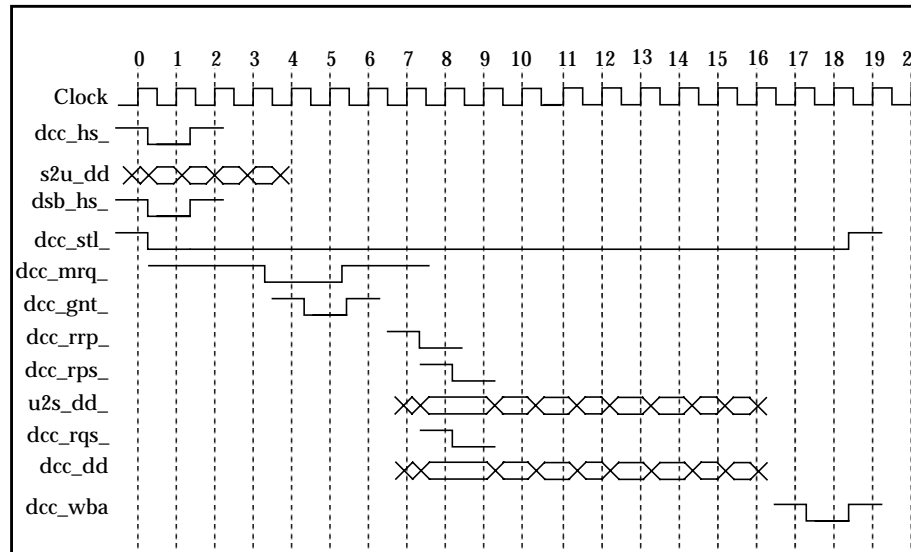


Figure 11-3 Servicing a Partial Write

Partial Write Cycles:

1. dcc_hs_ signals a partial write; data is clocked into the merge buffer; dsb_hs_ inserts the partial write into the DMA score-board which issues a Read-to-Own onto the UPA; the Merge Buffer asserts dcc_stl_ to stall all DMA activity; note that partial write data on s2u_dd varies from one to eight cycles.
2. The Read-to-Own is inserted into the S2U Header Fifo.
3. The Merge Buffer prepares to steer the MUXs to merge the data.
4. dcc_mrqr_ is asserted by the Merge Buffer to request the internal buses for a Writeback.
5. Since all DMA activity is stalled, the Merge Buffer wins arbitration.
6. Waiting for Read-to-Own to come back.

7. Waiting for Read-to-Own to come back.
8. `dcc_rrp_` indicates Read-to-Own data ready (note that UPA read latency varies and is more than the two cycles shown in this example).
9. The Merge Buffer asserts `dcc_rps_` to start taking the Read-to-Own data; it also asserts `dcc_rqs_` to inform the Bus Controller that it is starting to transmit the Writeback data; during the next eight cycles (including this cycle), the Merge Buffer is steering the appropriate MUXs to merge the Read-to-Own data with the Partial Write data and sending it to the UPA DMA Write Fifo; note that the merging is purely flow-through.
10. Writeback data.
11. Writeback data.
12. Writeback data.
13. Writeback data.
14. Writeback data.
15. Writeback data.
16. Writeback data.
17. Waiting for Writeback Ack.
18. `dcc_wba_` is asserted to indicate that the Writeback has completed on the UPA (the UPA write latency varies and is more than the one cycle shown in this example).
19. `dcc_stl_` is de-asserted to allow DMA activity to continue.

Note: Partial Write Data

Two different sources of partial data include SBus WSs and Streaming Cache flushes. In both cases, the data is contiguous. Data from the SBus is aligned and only of size 1,2,4,8,16, or 32 bytes. Data from the Streaming cache can be unaligned and of any size. The six LSBs of the address indicate the starting byte of the valid data. `stc_ep` is the end pointer and points to the byte after the last byte of valid data. This allows the Streaming Cache to flush partial lines with one transaction. Note that if the line is full (64 Bytes), it goes directly to the UPA as a Write-Invalidate.

12.1 Overview

The SBus Module (SBM) provides the functionality of the IEEE P1496 (SBus) while supporting communication between the UPA-64 and the SBus. The SBM supports six SBus Master/Slave Slots, as well as, one SBus Slave Only Slot.

The SBM communicates with the UPA interface for servicing PIO transactions, and communicates with the IOMMU, Streaming Cache, and the UPA interface for servicing DVMA transactions.

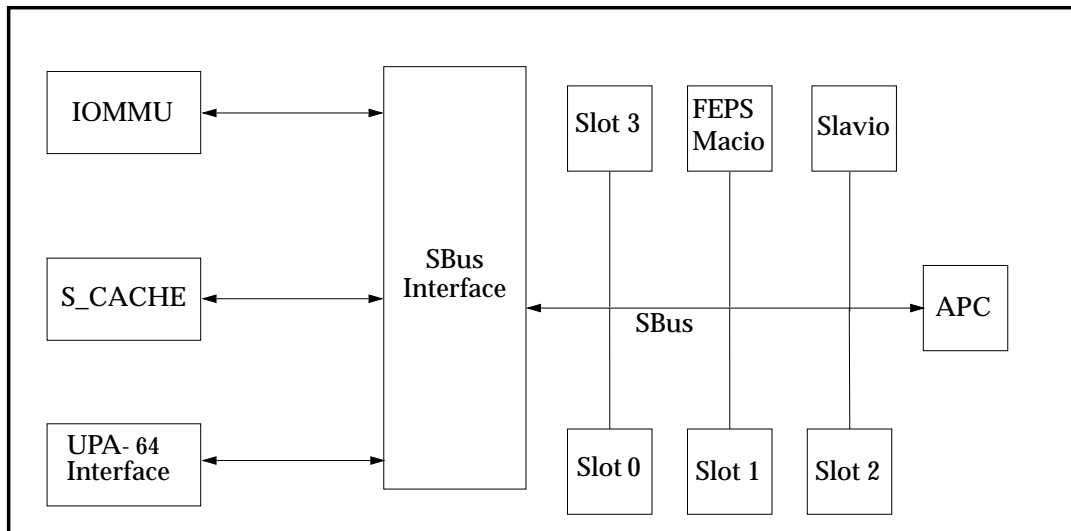


Figure 12-1 SBus Module Interface Diagram

12.1.1 Highlight of Features Supported

- SBus Master, SBus Slave, and SBus Controller functionality (DVMA, PIO, and Slot-to-Slot transfers)
- 6 Master/Slave slots, 1 Slave slot
- 32-bit and 64-bit Extended Transfer Mode
- SBus parity
- Consistent and Streaming Mode Access
- Round-robin SBus Arbitration
- Transfer sizes of 1, 2, 4, 8, 16, 32, and 64 bytes are supported
- One Slot Configuration Register per SBus Slot
- DVMA access to Slot Configuration Register
- DVMA Translation, Bypass, and Pass-through modes
- Burst to non-Burst PIO device transfers
- Slot-to-slot transfers on the same SBus supported; No Inter-SBus transactions are allowed (to remote SBus)
- Extended Mode Locks are not supported

12.1.2 Functional Overview

The main blocks in the SBM include the PIO controller, the DVMA controller, and the Address/Data path.

The PIO controller services all PIO transactions from the UPA to the SBus and to the SBM internal registers. For a given PIO transaction to a given slot, the PIO controller must check the slot's configuration register to see what PIO transaction is supported (64 bit mode support, burst support). Then the PIO controller issues the appropriate transaction on SBus, breaking the transaction down to smaller size transactions if necessary.

The DVMA controller services all DVMA transactions from SBus to the UPA and to another slot on the same SBus. It must communicate with the IOMMU for virtual-to-physical address translation and to the Streaming Cache, if the device accesses a page that is streamable. The DVMA controller also handles slot-to-slot transfers.

The Address/Data path contains the PIO RW Buffer to temporary store PIO data, the DVMA RW Buffer to temporary store DVMA data, the SBM registers, and all the appropriate muxes necessary to steer the address and data to and from the SBus.

The other blocks in the SBM include the following:

- Time-Out Counters: monitoring time-out on SBus
- Reset Block: controlling SBus and SBM resets
- Arbiter: arbitrating among PIO and DVMA requests to the SBM
- a miscellaneous block

A general block diagram of the SBM is shown in Figure 12-2, "SBM Block Diagram."

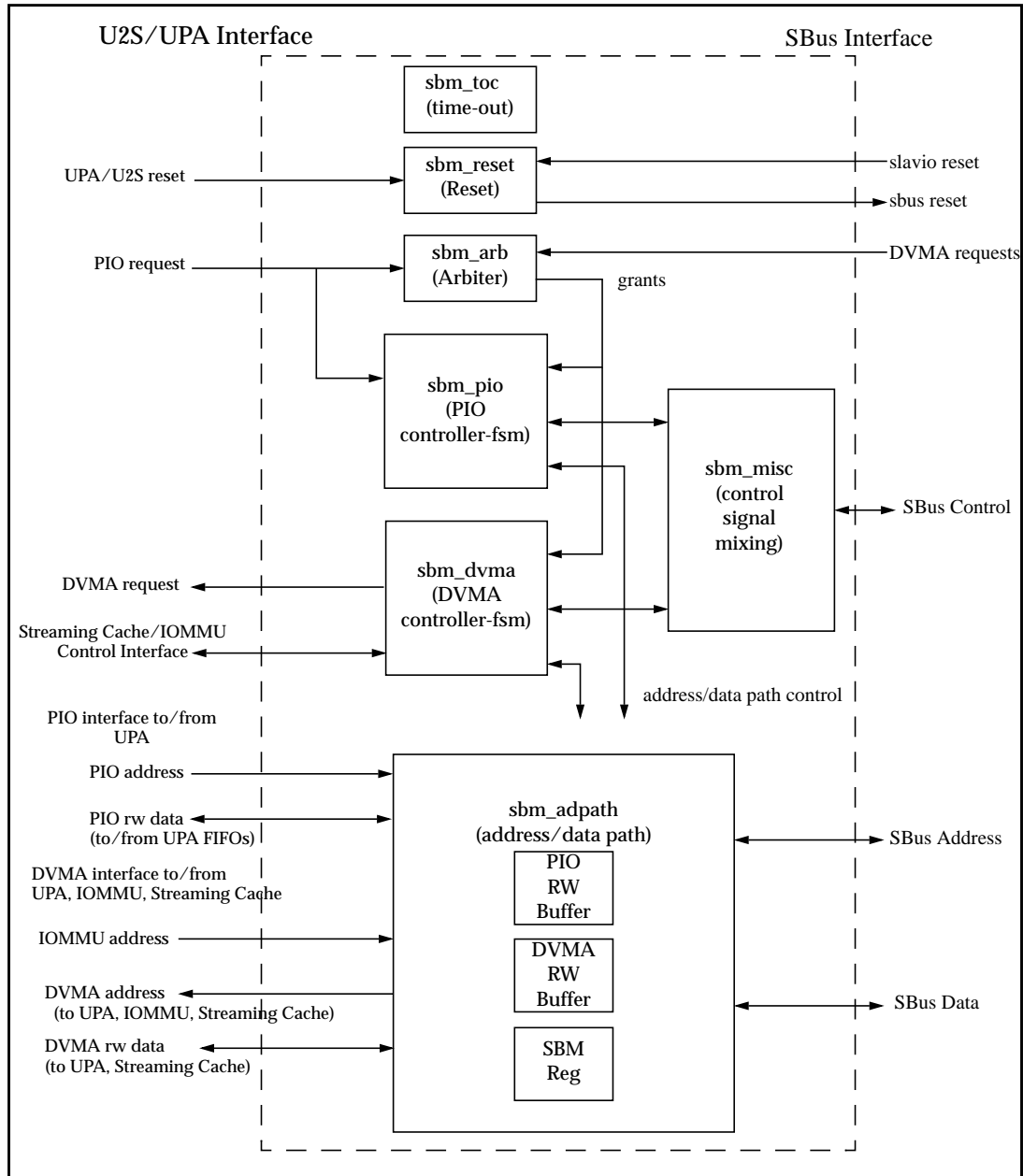


Figure 12-2 SBM Block Diagram

12.2 SBus Module Functional Description

The following section succinctly describes each block in the SBM. General policies and features are discussed.

12.2.1 Time-Out Counters (*sbm_toc*)

The Time-Out Counters monitor SB_AS_ (SBus Address Strobe) for the 255 cycle time-out time specified in the SBus Specification. If SB_AS_ is asserted for 255 SBus clocks, a time-out condition has occurred.

For PIO transactions, this will occur if the target slot has not responded within the time-out time. As a result, the SBM will Error Ack the slot and terminate the transaction (for example: - deassert SB_AS_ and release the SBus). For PIO Writes, the error is logged in the SBM AFSR and AFAR (Asynchronous Fault Status and Address Registers - see Chapter 4, “Programming Model”. For PIO Reads, the error will be reported back to the UPA as a P_RERR P_REPLY (Read Data Error).

For slot-to-slot DVMA transactions, like PIO transactions, this will occur if the target slot has not responded within the time-out time. The SBM will Error Ack the slot and terminate the transaction. There will be no log of the error. SBM’s policy of slot-to-slot DVMA transactions is that it is the responsibility of the master slot to handle errors.

For DVMA to memory, this will occur in only one case: a 64 bit mode write to the Streaming Cache. If the write page hits in the Streaming Cache or is a 64 byte write, then the SBM will speculatively assert SB_AS_, assuming that the Streaming Cache can accept the data. However, if the following, extremely rare, conditions also exist, then it is possible for a time-out condition to occur: the write is a page hit line miss in the Streaming Cache, or it is a 64 byte write; and the Streaming Cache flush buffer is full; and the UPA is saturated. In this extremely rare case, the Streaming Cache cannot accept the data since its flush buffer is full and the UPA is so saturated that it cannot service any transactions from U2S for 255 SBus clocks. When this time-out condition occurs, the SBM will re-run the slot. This is the only case where SBM re-runs a master.

For a summary of error handling, see Section 12.3, “Summary of Error Handling in SBM.”

12.2.2 Reset (*sbm_reset*)

There are two important resets controlled by the SBM: the SBM reset and the SBus reset.

The SBM reset controls when the entire SBM wakes up from reset and begins processing transactions. This reset is deasserted when both U2S reset and Slavio reset are deasserted.

The SBus reset is the actual reset on the SBus. It is deasserted when U2S reset is deasserted, which is based on UPA reset.

This reset scheme is required to prevent SBM from coming out of reset until Slavio is ready. The typical reset sequence is as follows:

1. U2S Reset deasserted.
2. SBus Reset deasserted by SBM.
3. Slavio Reset deasserted by Slavio.
4. SBM now ready.

12.2.3 Arbiter (*sbm_arb*)

The Arbiter is a round-robin arbiter that arbitrates among seven sources: six DVMA master slots, and one UPA PIO request. Only one transaction is allowed to proceed at a time regardless if it is PIO or DVMA. This includes PIO to the SBM registers.

The Arbiter inserts one extra dead cycle between any DVMA request followed by any PIO request. A dead cycle is defined by SB_AS_ being deasserted for two cycles instead of one. This is to prevent a bus contention of a DVMA Read followed by a PIO Write (shown in the figure below). This also makes the SB_AS_ assertion time consistent across PIOs and DVMA.

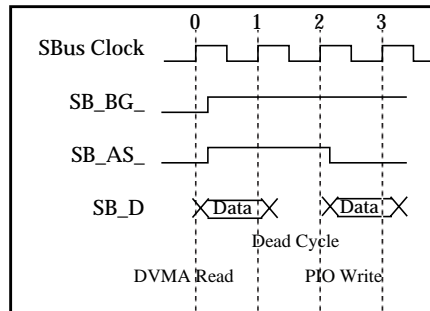


Figure 12-3 SBus Arbiter Dead Cycle

12.2.4 PIO Controller (*sbm_pio*)

The PIO Controller services PIO transactions from the UPA in a FIFO manner. Only 1 PIO transaction is serviced at a time; no PIO transaction can bypass the one before it. PIO data is given/taken directly to/from the UPA data FIFOs with an intermediate 8 byte PIO RW buffer (see Address/Data Path).

A slot is assumed to support 32 bit mode PIO transfers of 1, 2, or 4 bytes. If a slot supports more than this, the appropriate bits in the slot's configuration register must be set (64 bit mode, 8 byte bursts, 16 byte bursts, 32 byte bursts, 64 byte bursts, parity).

For a given PIO size transfer, the appropriate burst size bit must be set in the configuration register, else the transfer is broken down in the next lowest supported size. The SBM automatically increments the address and provides the appropriate data for each transaction.

Table 12-1 "Burst to Non-Burst Examples," lists some examples (X signifies don't care).

Table 12-1 Burst to non-Burst Examples

| PIO Transaction Size | ETM | 8 | 16 | 32 | 64 | Resultant Transaction(s) on SBus |
|----------------------|-----|---|----|----|----|--|
| 8 bytes | 0 | 1 | X | X | X | one 32 bit mode 8 byte transaction |
| 8 bytes | 0 | 0 | X | X | X | two 32 bit mode 4 byte transactions ^[1] |
| 16 bytes | 0 | 1 | 0 | X | X | two 32 bit mode 8 byte transactions |
| 8 bytes | 1 | 0 | X | X | X | two 32 bit mode 4 byte transactions ^[1] |
| 8 bytes | 1 | 1 | X | X | X | one 64 bit mode 8 byte transaction |
| 64 bytes | 0 | 0 | 0 | 0 | 0 | 16 32 bit mode 4 byte transactions ^[1] |
| 64 bytes | 1 | X | X | X | 1 | one 64 bite mode 64 byte transaction |

1. Each 32 bit mode 4 byte transaction can also be dynamically sized down to multiple 1 or 2 byte transactions by the slave

When a transaction is broken down into smaller transactions, after each 8 bytes of transfer, an additional dead cycle is inserted (SB_AS_ is deasserted for two cycles as opposed to one). So, if a 32 byte transfer is broken down into eight 32 bit mode 4 byte requests, there will be a dead cycle after the 2nd, 4th, and 6th 4 byte transaction. This is because the PIO RW buffer is only 8 bytes and it takes U2S an extra cycle to move the data to/from the buffer to the UPA buffers.

Also note that the PIO transaction from the UPA is kept “semi-atomic” on the SBus. That is, even if the PIO transaction is broken down into multiple PIOs on the SBus, they are all performed in sequence without re arbitration, allowing the entire data to be transferred before another PIO or DVMA is allowed to take place. The exception to this is if the slave re-runs the SBM, at which time, the SBus is released and goes through re arbitration. At this time, DVMA can occur before the PIO completes.

Each time a slave reruns the SBM, the SBus is released and the PIO transaction goes through re arbitration for the SBus. If there are no other requests, then the exact same PIO transaction will be tried again, immediately. To avoid bus contention, similar to the Arbiter dead cycle condition), an extra dead cycle (SB_AS_ is deasserted for two cycles as opposed to one) is inserted between a PIO Read transaction and a PIO Write transaction as shown below:

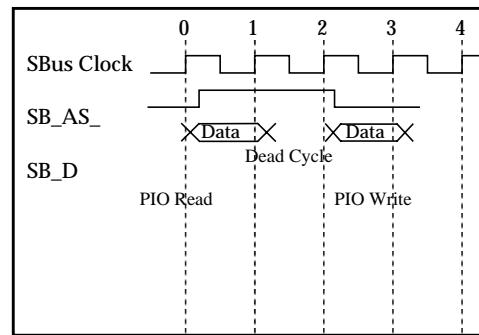


Figure 12-4 PIO Dead Cycle

PIO Error Conditions

For PIO Writes, if an Error Ack or an invalid Ack is received, an interrupt is sent and the error is logged in the SBM AFSR and AFAR.

For PIO Reads, if an Error Ack or an invalid Ack is received, the error is sent back to the UPA as a P_RERR P_REPLY. If the data received from the SBus has a parity error and the parity check bit is set in the slot configuration register, then the data's ECC is corrupted with a UE out to the UPA. In addition, the error is logged in the SBus Control Register.

For all PIO transactions, a late error causes an interrupt to be sent and is logged in the SBM AFSR and AFAR.

For PIO time-outs, see Section 12.2.1, "Time-Out Counters (sbm_toc)."

For a summary of error handling, see Section 12.3, "Summary of Error Handling in SBM."

12.2.5 DVMA Controller (*sbm_dvma*)

The DVMA Controller services DVMA request from SBus to the UPA or to another slot on the same SBus. Since the mode of transfer (64 bit mode, size) is specified by the device, there are no configuration bits needed for DVMA. The SBM will process the request from the slot no matter what size and mode is requests. SBM supports both 32 and 64 bit mode of operation, and burst sizes of 1, 2, 4, 8, 16, 32, and 64 bytes. 128 byte bursts are not supported.

The DVMA Controller services one DVMA request at a time. For each DVMA read or write, the SBM will remain connected with the master until the transfer is completed. Note that for reads, this includes the latency to memory and back. The only case when the DVMA Controller will Rerun the master is explained in Section 12.2.1, "Time-Out Counters (*sbm_toc*)."

Transaction Types

There are several types of DVMA, as listed below:

1. Bypass - Consistent.
2. Pass-through - Consistent.
3. Slot Configuration Register Access.
4. Translation - Streaming.
5. Translation - Consistent.
6. Translation - Local.
7. Translation - non-Cacheable.

Note the following about the preceding DVMA types:

- Types 1, 2, and 5 are Consistent transactions which means that the data is transferred to/from memory (cacheable space) only. Consistent transfers use the DVMA RW ping-pong buffers (see Address/Data Path)
- Type 3 allows the slot to directly read and write its own Slot Configuration Register
- Translation type DVMA accesses (4, 5, 6, and 7) require the IOMMU or the Streaming Cache, or both, to specify which translation type
- Type 4 is a Streaming transaction which means that the data is transferred to or from the Streaming Cache
- Type 6 is a local SBus slot to another slot transfer
- Type 7 is a transfer to I/O space on the UPA, such as to a frame buffer

When a DVMA transaction is received, the IOMMU Enable Bit, the Slot Configuration Bypass Bit, and the Virtual Address determine if the transaction is of the first three types (Bypass, Pass-through, Slot Configuration Register Access - see the Ultra SPARC System Specification for more details). If this is the case, then the physical address (or target register) is formed directly from the virtual address and the Slot Configuration Register. The IOMMU and Streaming Cache are not involved in these types of transactions.

If the transaction is not of the first three types, then the DVMA controller signals both the IOMMU and the Streaming Cache to begin their lookups. The Streaming Cache's result always takes precedent over the IOMMU's result. So, if the Streaming Cache responds with a hit, then the transaction will proceed as a Streaming Access (type 4). If the Streaming Cache responds with a miss, then the results of the IOMMU are used.

If the IOMMU responds with a miss, then a table walk is initiated and the DVMA transaction stalls until the IOMMU responds with the proper translation after the table walk. The IOMMU supplies the physical address as well as the type of transaction (4 - streaming, 5 - consistent, 6 - local, or 7 - non-cacheable).

It is possible for the following events to occur: IOMMU responds first with a miss, SBM tells IOMMU to start table walk, Streaming Cache then responds with a hit, DVMA transaction begins and completes to the Streaming Cache. In this case, the IOMMU is still outstanding with a table walk. The SBM will then hold the SBus by keeping SB_BG_ asserted after the transaction has completed until the IOMMU responds, at which time the SBM will release the SBus and the next transaction can occur.

DVMA Error Conditions

- For any translation error (such as: write to a read-only page, IOMMU table walk miss), the SBM will Error Ack the master and terminate the transaction
- For a parity error on the virtual address or ETI of a transaction (if the Parity Enable bit is set), the SBM will Error Ack the master and terminate the transaction
- The following error conditions apply only to DVMA from the SBus to the SBM (for example: not local slot-to-slot transfers)
- For DVMA writes, if the data has a parity error and if the Parity Enable bit is set, the SBM will send a late error to the master, corrupt the ECC of the data going to UPA with a UE, and log the error in the SBM Control Register
- For DVMA reads, if the data coming from the UPA has a UE, the SBM will give an Error Ack to the master corresponding to the corrupted data cycle and terminate the transaction
- For local slot-to-slot transfers, only the time-out condition is monitored. See Section 12.2.1, "Time-Out Counters (sbm_toc)." No other error checking is performed. It is the responsibility of the master slot to "know" which modes and sizes the slave slot supports; and it is the responsibility of the slave slot to handle any and all errors
- Also see Section 12.2.1, "Time-Out Counters (sbm_toc)," for the only time the SBM will give a DVMA master a Rerun Ack

12.2.6 Address/Data Path

The following is a block diagram of the Address/Data Path.

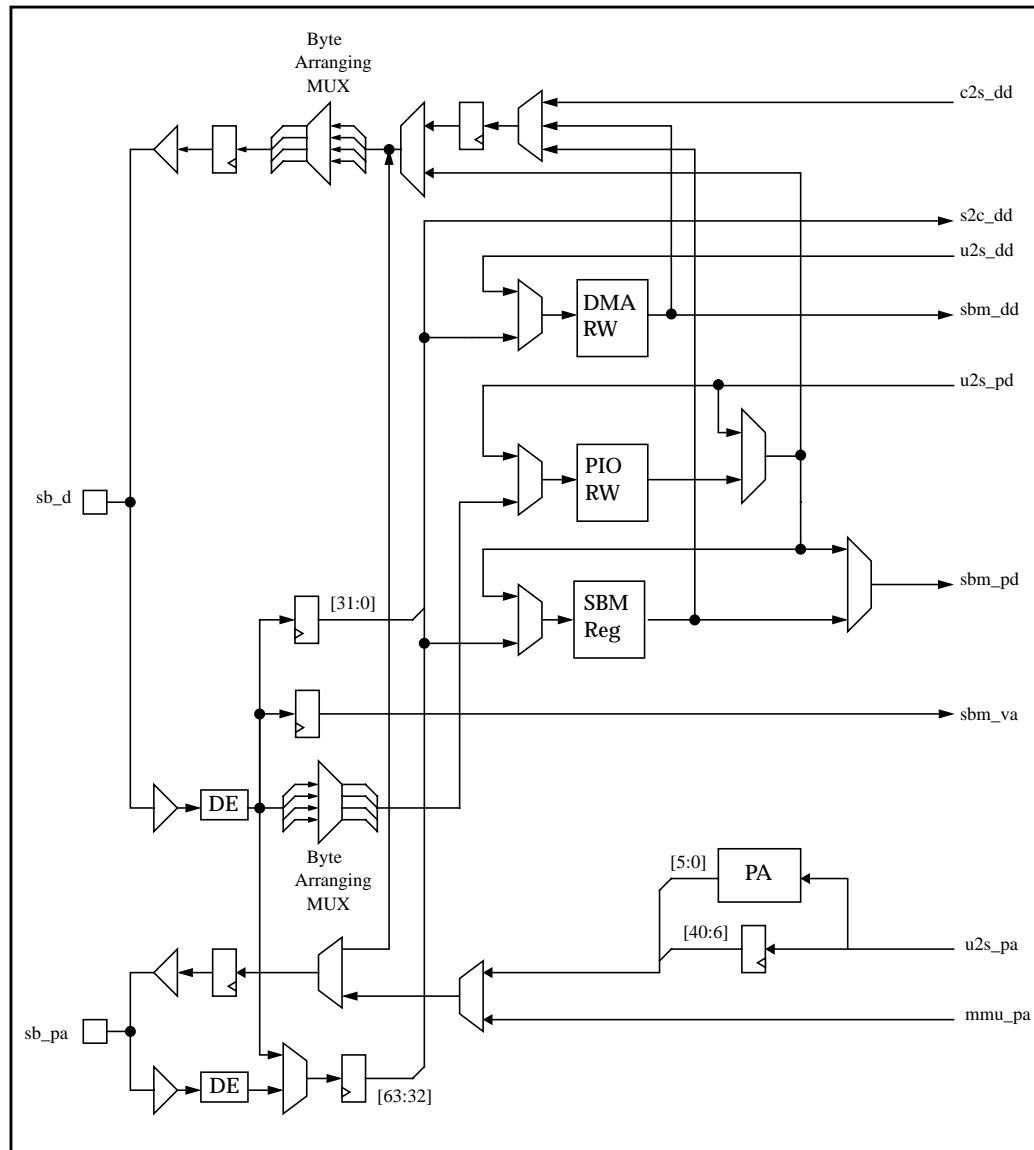


Figure 12-5 Address/Data Path Block Diagram

The byte arranging MUXs are for PIO transactions only. For 1 byte PIO Writes, the data is duplicated four times to fill the 32 bits. For 2 byte PIO Writes, the data is duplicated two times to fill the 32 bits. For PIO Reads, the byte arranging mux is used to pick up the data in the appropriate byte locations.

For DVMA, the SBM always responds with a 4-byte Ack for 32 bit mode transactions, and an 8-byte Ack for 64 bit mode transactions, so byte arranging MUXs are not required.

The PIO RW Buffer is an 8 byte buffer used to hold PIO Read and Write data.

The DVMA RW Buffer is a ping-pong buffer composed of two independent 64 byte buffers. They are primarily for improving the performance of consistent DVMA writes. As one buffer is emptying to the UPA, the other buffer can accept another DVMA write from the SBus.

Interrupt Ordering

The order of Interrupts and DVMA is guaranteed for non-streaming accesses. For DVMA Writes, data may sit in the DVMA RW Buffers while the device asserts its interrupt. The mechanism for insuring order is handled by a simple handshake between the SBM and the MDU (Mondo Interrupt Unit). The MDU will assert the signal `sbm_iv_` to inform the SBM of a pending interrupt. After the DVMA RW Buffer is emptied, the SBM then asserts `sbm_ord_` to inform the MDU to go ahead with the interrupt.

12.2.7 Other SBM Notes

The Fast SBus bit in the SBM Control Register affects the timing of PIO Reads only. If this bit is clear, when the U2S is running slow compared to the UPA, then the SBM will delay the `sbm_eot_` signal to the UPA until the last data is processed. Otherwise, the `sbm_eot_` signal is given to the UPA two cycles before the last data is ready, allowing the UPA to give an earlier `P_REPLY`.

Power management in the SBM is implemented with DVMA. During Power Down mode, if a master asserts its request (`SB_BR_`) and the appropriate Arbiter Enable bits are set (`ARB_EN` in SBus Control Register), then the SBM will raise an interrupt to wake up the system. The requesting master is not granted until the system clears the `WAKEUP_EN` bit in the SBus Control Register (see the Ultra SPARC System Specification for details).

12.3 Summary of Error Handling in SBM

The following table summarizes the error handling policies of the SBM.

Table 12-2 Summary of Error Handling in SBM

| Error | Action(s) |
|-------------------------------------|---|
| PIO Read Time-out | Error Ack the slave, terminate cycle, P_RERR P_REPLY |
| PIO Write Time-out | Error Ack the slave, terminate cycle, log in SBM AFSR/AFAR |
| DVMA Slot-to-slot Time-out | Error Ack the slave, terminate cycle |
| PIO Write Error or Invalid Ack | send interrupt, log in SBM AFSR/AFAR |
| PIO Read Error or Invalid Ack | P_RERR P_REPLY |
| PIO Read Parity Error | UE on data to UPA, log in SBus Control Register |
| PIO Late Error | send interrupt, log in SBM AFSR/AFAR |
| DVMA Write Parity Error | send later error to master, UE on data to UPA, log in SBM AFSR/AFAR |
| DVMA Read UE | Error Ack master |
| DVMA Translation Error | Error Ack master |
| DVMA Parity Error on Address or ETI | Error Ack master |

13.1 Overview

The SBus IOMMU performs virtual-address-to-physical-address translation during DVMA cycles. The SBus master devices provide a 32-bit virtual address at the beginning of DVMA transfers. The IOMMU translates it into 41 bits of physical address.

The IOMMU consists of a Translation Look-Aside Buffer (TLB), a Translation Storage Buffer (TSB), and a software-managed data structure. Sixteen entries of fully-associative TLB are implemented in the U2S. TSB is a second-level lookup table which resides in memory. Hardware performs the TSB lookup when the translation cannot be found in the TLB. An error is returned to the device if the TSB lookup fails to locate a valid mapping. The software managed data structure provides information during the setup of the TSB table entry, see Chapter 4, “Programming Model,” Table 4-21, “IOMMU Translation Table Entry (TTE).”

The SBus IOMMU supports two different page sizes, 8K and 64K. Mixed page sizes can be used in the system, but the TSB table lookup only assumes the smaller page size. No overlapping of pages is allowed. Bypass operation is supported to allow devices having a translation facility to bypass IOMMU.

13.2 Mode of Operations

Depending on the value of the MMU_EN bit of the IOMMU control register, the BY bit of SBus Slot Configuration Register, and the SBus virtual address bits [31:30], the IOMMU is put into different operating modes as shown in Table 13-1.

Table 13-1 IOMMU Mode of Operation

| MMU_EN | BY | VA[31:30] | Mode |
|--------|----|-----------|------------------------------------|
| X | 1 | 00 | Bypass |
| X | 1 | 01 | Slot configuration register access |
| 1 | 1 | 1X | Translation |
| 0 | 1 | 1X | Pass-through |
| 0 | 0 | XX | Pass-through |
| 1 | 0 | XX | Translation |

13.2.1 Translation Mode

The IOMMU is in translation mode if a translation is enabled, MMU_EN bit is set, and the device performing transfers is not using bypass mode. When a translation is started, IOMMU hardware will perform TLB lookup first. If a TLB miss happens, the hardware automatically starts the TSB lookup. If the TSB lookup locates a valid mapping for the virtual page, information in the TSB entry will be loaded into the TLB and the translation continues. If the TSB lookup results in a miss, an error will be returned to the SBus master.

The virtual address consists of two fields: virtual page number and page offset. Page offset stays the same from virtual address to physical address. SBus IOMMU supports two page sizes, 8K and 64K. The conversion of virtual address to physical address are shown below.

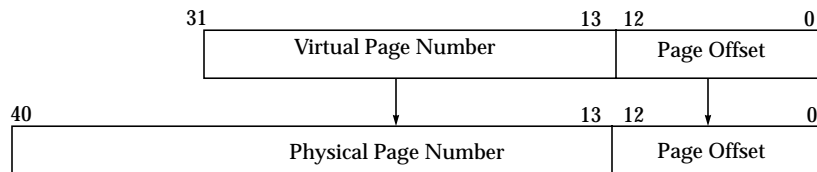


Figure 13-1 Virtual-to-Physical-Address Translation for 8K Page Size

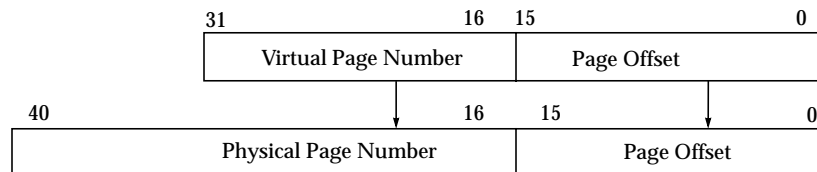


Figure 13-2 Virtual-to-Physical-Address Translation for 64K Page Size

13.2.2 Bypass Mode

The implementation of the IOMMU allows the SBus device have its own MMU and bypass the IOMMU supported by the system. A device is operating in bypass mode if the BY bit in its slot configuration register is set and the high order two virtual address bits, VA[31:30], are set to 00.

Although a device in bypass mode has access to the full physical address space, it has direct access to only one gigabyte of physical address space. To access the 41-bit physical-address space of the system, high order address bits, PA[40:30], come from the SBus Slot Configuration Register (SSCR) SEGA[40:30]. The physical address is formed by concatenating SEGA[40:30] and VA[29:00]. If the DMA access is crossing the one Gigabyte boundary, the device needs to change the SEGA[40:30] to point to the next one gigabyte segment. The SEGA[40:30] is writable by device through the DVMA access as shown in Chapter 4, "Programming Model," Table 4-17, "SBus Slot Configuration Register (per Slot)."

A device in the 41-bit physical address space can be either cacheable or non-cacheable. The CP bit of the SSCR controls whether a coherent transaction should be issued to the UPA interconnect. Software should set the CP bit properly in order to avoid misbehavior of the hardware.

13.2.3 Pass-Through Mode

When the IOMMU is disabled and the DVMA transfer is not in bypass mode or accessing the SSCR, the access is considered to be in pass-through mode. Pass-through mode allows access to the lower one or four gigabytes of memory address space only. The high order 10 bit or 9 bit of the physical address will be padded with 0. A DVMA access in pass-through mode will always cause a coherent transaction to the UPA interconnect.

13.3 Translation Storage Buffer

Translation Storage Buffer, or TSB, is a translation table in memory. It contains mapping information for the virtual pages. IOMMU hardware will look up this table if a translation cannot be found in the TLB. Each entry in the table takes eight bytes.

Several TSB table sizes are supported in the system. The size of TSB table is specified by the TSB_SIZE field of IOMMU control register. Table sizes supported are 1K, 2K, 4K, 8K, 16K, 32K, 64K, and 128K entries. Software must set up the TSB before it allows translation to start.

13.3.1 Translation Table Entry (TTE)

Each entry in the TSB table is called a Translation Table Entry, or TTE. The entry contains translation information for a virtual page. The IOMMU hardware reads in the translation information from the TTE during TLB miss. The “VALID” bit in the entry must be set to contain valid information. Information stored in the TTE is shown in Table 13-2.

Table 13-2 TTE Data Format

| Field | Bits | Description |
|-------------|---------|---|
| DATA_V | [63] | Valid bit, indicating the TTE entry has valid mapping |
| DATA_SIZE | [61] | Page size of the mapping 0 = 8K, 1 = 64K |
| STREAM | [60] | Set if the page is streamable (see Chapter 14, “Streaming Cache,” for information on the stream mode) |
| LOCALBUS | [59] | Set if the physical address points to local the SBus |
| DATA_SOFT_2 | [58:51] | Reserved for software use |
| DATA_PA | [40:13] | Contains bits [40:13] of the physical address, some of the low order bits are not used for the 64K page |
| DATA_SOFT | [12:7] | Reserved for software use |
| CACHEABLE | [4] | Set if this page is mapped cacheable |
| DATA_W | [1] | Set if this page is allowed to be written |

13.3.2 TSB Lookup

During the TSB lookup, the physical address for the TTE entry is formed based on the following information:

- base address of the TSB table
- assumed page size during TSB lookup, TBW_SIZE of the IOMMU Control Register
- the size of TSB table

The TSB Base Address Register contains the physical address of the first TTE entry in the TSB table. The table must be aligned on an 8K boundary regardless of table size. The lower-order 12 bits of this register are assumed to be 0. The physical address for the entry in the TSB table is formed by adding the base address and offset together.

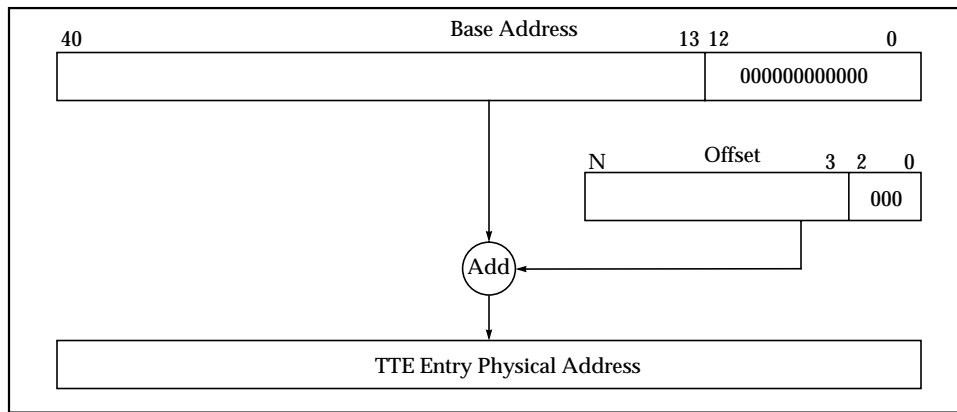


Figure 13-3 Computation of TTE Entry Address

Offset of the TSB table is calculated based on a direct mapped TSB table. Table 13-3 shows how the offset is obtained. The lower order three bits is set to 0x0 because each entry is eight bytes in size.

Table 13-3 Offset to TSB Table

| TSB Table Size | N | Offset (8K TSB Lookup Page Size) (TBW_SIZE=0) | Offset (64K TSB Lookup Page Size) (TBW_SIZE=1) |
|----------------|----|---|--|
| 1K | 12 | [VA[22:13], 000] | [VA[25:16], 000] |
| 2K | 13 | [VA[23:13], 000] | [VA[26:16], 000] |
| 4K | 14 | [VA[24:13], 000] | [VA[27:16], 000] |
| 8K | 15 | [VA[25:13], 000] | [VA[28:16], 000] |
| 16K | 16 | [VA[26:13], 000] | [VA[29:16], 000] |
| 32K | 17 | [VA[27:13], 000] | [VA[30:16], 000] |
| 64K | 18 | [VA[28:13], 000] | [VA[31:16], 000] |
| 128K | 18 | [VA[29:13], 000] | Not allowed |

The TBW_SIZE should be set to 0 if there is an 8K page size or mixed 8K and 64K page sizes are used for DVMA mappings. Each 64K page will use up eight entries of TTE if the TBW_SIZE is set to 0. Software must fill all eight entries with the same information if a mixed page size is used.

13.4 Translation Errors

Errors detected by the IOMMU during translation are reported to the SBus master device. It is up to the SBus device to report the error. Information associated with the erroneous transaction should also be kept in the SBus device. No system resources are allocated for this purpose. Errors detected by IOMMU are invalid error and protection error. An invalid error happens if the TTE read by the IOMMU hardware is an invalid entry. A protection error happens when a device is trying to write to a page that has no write permission. Both errors are reported through SBus error acknowledge.

13.5 IOMMU Demap

After the mapping between virtual address and physical address is established, any change to the mapping information needs to demap the existing mapping before new mapping can be used by the device. Demap is required for the following occasions: taking down existing mapping to make physical memory available to other virtual addresses or changing access permission to a page.

During IOMMU demap, the SBus device is not allowed to use the page that is being demapped. If a device tries to access a page that is being demapped, unexpected results may happen. The following events are needed to demap a page in the IOMMU.

- Flush the streaming cache if the page is marked streamable
- Update the proper TSB entry with new information
- Perform TLB flush with the virtual page number

TLB flush is initiated by writing to the IOMMU Flush Address Register with specified virtual page number. Match criteria are different for 8K and 64K page sizes. Hardware performing the flush will adjust match criteria based on the page size. The matched entry in the TLB will be marked invalid.

13.6 TLB Initialization and Diagnostics

The IOMMU provides direct access to its internal resources, such as TLB Tag, TLB Data, LRU Queue, and Match Comparison Logic: see Chapter 4, "Programming Model," Section 4.5, "IOMMU Registers."

After the power is turned on the contents of IOMMU is undefined. Before any DVMA is allowed to use the IOMMU, all TLB entries need to be invalidated. This is done by writing 0 to the V bit in every entry of TLB Data RAM.

14.1 Overview

The streaming cache implemented in the U2S is a small-size fully-associative cache managed by hardware to accelerate SBus DVMA to and from memory. A DVMA page can be mapped in consistent or stream mode. Only the page mapped streamable DVMA is allowed to use the streaming cache.

The stream DVMA needs software maintenance. These include page invalidation, page flush, and flush synchronization. Certain restrictions are necessary to ensure the correctness of data and proper performance of the streaming cache.

14.2 Consistent DVMA and Stream DVMA

DVMA to and from memory can be operated in stream mode or consistent mode. The operating mode is controlled by the S bit of the IOMMU mapping. Since the stream property is controlled solely by the IOMMU mapping, the property is assigned based on the page. It is not possible to have stream and consistent mode assigned to a page at the same time. The stream and consistent property can be changed from time to time, but software needs to take proper action during demap. This will be described in a later section.

14.2.1 Consistent DVMA

Consistent DVMA participates in cache coherence and does not make use of the streaming cache. It is useful for IOPBs and cases where the overhead of managing the streaming cache outweighs the benefits. Consistent DVMA are used during the following DVMA transfers.

- DVMA transfer to and from memory mapped in consistent mode

- DVMA transfer to and from memory using IOMMU bypass or pass-through mode
- DVMA transfer to and from non-cacheable UPA address space

14.2.1.1 Consistent DVMA Write

DVMA writes to memory in consistent mode will be observed in issuing order at the coherence domain. Write buffers are provided in the U2S for consistent write operation. No synchronization mechanism is needed for the consistent DVMA write buffers because interrupt is guaranteed not to surpass the previous consistent write to the UPA interconnect and write ordering is maintained. No order of completion is guaranteed between consistent DVMA write and the following operations.

- Stream write
- SBus DVMA to SBus device
- PIO accesses

All consistent DVMA writes will be observed at the UPA interconnect. Since the unit of coherency at UPA interconnect is 64 bytes and the U2S does not implement cache at the UPA domain, any consistent DVMA partial write of less than 64 bytes needs to be performed with the following operations in their respective order.

1. Ownership read from coherence domain.
2. Merge write data with returned data.
3. Write data into memory.

Once the U2S gets temporary ownership of the cache line, any access to the same cache line will be blocked by the USC until the memory write is completed. DVMA partial write to memory in consistent mode is discouraged because of the expensive operations.

14.2.1.2 Consistent DVMA Read

During consistent DVMA read, the U2S takes a snapshot of the data from the coherence domain and returns the data to the SBus device. The U2S will issue a read-to-discard UPA transaction to get the data without the transfer of ownership. The read will always be 64 bytes in size. Only requested data will be returned to the SBus device, remaining unused data will be thrown away.

Data returned to the SBus master will become stale if a UPA master updates the same memory locations with the new value. To get the latest data, the SBus device needs to perform another DVMA read.

14.2.2 Stream DVMA

The U2S provides 16 64-byte lines of read-ahead and post-write cache to speed up stream-mode DVMA. The allocation of cache line is based on one line per virtual page of 8K bytes. Although IOMMU can map a page to be 8K or 64K bytes, the streaming cache will assume smaller size mapping. The allocation is managed by hardware using the LRU algorithm.

Data transferred in stream mode may not be observed in issuing order. It also needs software intervention to help force data into the coherence domain. Only DVMA access to and from system memory is allowed to use stream mode.

14.2.2.1 Stream Read

When the SBus device starts a DVMA read, the U2S will use the SBus virtual address to look up the streaming cache and IOMMU at the same time. If the access results in a hit on the cache line, data will be provided directly from the streaming cache. Otherwise, a line will be allocated to the page if not already done, and data will be fetched from the coherence domain and filled into the streaming cache. This saves the latency and reduces the consumption of UPA/memory bandwidth of having to get data from the coherence domain on every DVMA read.

To further enhance performance, the streaming cache will perform pre-fetch when it senses that a current DVMA operation is going to consume the last byte of data in the cache line. However, if the address is going to cross a page boundary, prefetch is inhibited. The prefetch capability is transparent to software.

Once the data gets into the streaming cache, it will not remain coherent with the rest of the system. Data can be overwritten at the coherence domain without the notice of the streaming cache and therefore the DVMA master. The software needs to make sure pages marked streamable are not sensitive to this characteristic.

14.2.2.2 Stream Write

The streaming cache acts as an accumulating write buffer for the DVMA write. DVMA writes, with a sequentially-increasing address, will be accumulated in the cache line. When the write reaches the end of a cache line, a flush operation will be initiated. Doing this saves the expensive operations of a DVMA partial write to memory.

The side-effect of buffering write is that the data may stay in the streaming cache for a long period of time if no software mechanism exists to force the data out. Flush and flush synchronization are needed to ensure that the consumer of data gets the most updated information.

14.3 Streaming Cache Management

To ensure data consistency across the system, software needs to manage the streaming cache. Operations managing the streaming cache include invalidation, flush, and flush synchronization. Invalidation and flush are done by the same command. It involves the PIO write to the streaming cache Page Invalidate/Flush Register with a virtual page number provided as the PIO write data. The entry with the matching virtual page number will be invalidated if the page is clean, or flushed/invalidated if the page contains dirty data. Eight invalidate or flush operations are needed for a 64K page since the tagging occurs on only 8K boundaries and it is possible to have all eight pages present in the cache.

14.3.1 Streaming Cache Invalidation

The streaming cache needs to be invalidated in the following scenarios.

- *Before a device starts its first DVMA read:* A cache line may be brought into the streaming cache long before the device starts the DVMA read. If the memory location is updated after the line is in the cache, the device is not able to see the updated data
- *When a page is demapped:* The streaming cache also stores mapping information to speed up the flush operation. The mapping information is also used during prefetch. When a page is demapped, the mapping information in the streaming cache needs to be invalidated also

14.3.2 Streaming Cache Flush

Flush can be triggered by several sources as listed below. The first four in the list are invisible to the software.

1. *End of line flush:* When a DVMA write reaches the end of a cache line, the hardware will perform a line flush if the flush buffer is available. Otherwise, the line will stay in the cache until it is bumped out.
2. *Non-sequential write to the same line:* Any non-sequential write to a cache line will cause the existing line to be flushed before new data can be accepted.
3. *Line eviction on the same page:* If a line is partially filled and the device starts writing to a new line on the same page, buffered data for the previous write has to be flushed before new data can be accepted.
4. *Line eviction different page:* This happens when all the cache lines are used up and when the incoming DVMA accesses a page with no line allocated to it. If the LRU line has dirty data, it needs to be flushed before the line can be allocated to a new page.
5. *Software-triggered flush:* Software needs to flush the streaming cache at the end of the DVMA transfers. A software-initiated flush will never take longer than 0.5 seconds in a properly working system. A time-out of this duration can be used to recover from an undetectable hang.

To make sure all previous flush operations are completed at the coherence domain, the U2S provides a mechanism to synchronize the flush operation. The flush synchronization involves a PIO write to the streaming cache Flush Synchronization Register with the physical address of the flush flag provided as PIO write data. Only one write of the synchronization register is required as a barrier for all previous flush/invalidate writes.

The following is a sequence of events for performing flush and synchronization:

1. Grab the lock of the flush flag.
2. Initialize (in memory) the flush flag to 0x0.
3. Flush dirty pages.
4. Synchronize flushes.
5. Poll the flush flag (located at the PA contained in the flush synchronization register) until it becomes 0x1.
6. Release the lock of the flush flag.

14.4 Streaming Cache Error Handling

There is no difference in reporting errors between stream DVMA and consistent DVMA. The reader should consult the Error Handling section for more details on error reporting. The streaming cache does not keep any error state and therefore does not need software intervention for error management.

14.5 Software Notes

Certain limitations are imposed on the streaming cache. Some of them are for performance reasons and some may cause misbehavior of the system. They are listed below:

- To avoid unnecessary flushes, DVMA write should use an increasing sequential address. Using a decreasing sequential address will cause a flush operation on every DVMA write. Jumping around lines on the same page or jumping back and forth on the same line will increase the frequency of flushes on the DVMA write and unnecessary prefetch during DVMA read. All these should be avoided
- Avoid address wrapping on DVMA streaming writes
- To amortize the overhead of flush, it is recommended to use stream DVMA in transferring a large amount of data instead of small chunk of data
- When possible, a larger burst size should be used to maximize performance
- The streaming cache is tagged with virtual address. It does not provide alias detection on the mappings. Two devices may have different virtual pages that are mapped in the same physical page. If both are writing to the same physical memory location, or if one device is reading and the other is writing to the same memory location, the results of the operations are not guaranteed
- If the same physical page is allocated to two different DVMA devices and their physical address spaces are not overlapped, aliasing can be used to avoid unnecessary flushes due to interleaving accesses between two different lines on the same page

This chapter describes the Mondo Dispatch Unit (MDU). The MDU handles interrupts from SBus and the U2S internal sources.

15.1 Definition of Terms

Trap Global registers: A dedicated set of registers provided in UltraSPARC- 1 for servicing interrupts.

Mondo Vector: An interrupt request packet containing three 64-bit fields: PC, DATA (PTR), and DATA.

- PC: Program Counter for the interrupt service routine
- DATA: 2 data field associated with the interrupt. These can be defined to include the address of the transaction that erred, the contents of a status register. The U2S does *NOT* send any data with interrupts

ISR: Interrupt State Register

INR: Interrupt Number Register

OBIO: On-board IO devices

Prefixes to Signal Names

U2S - UPA to SBus buses (address and data)

S2U - SBus to UPA buses (address and data)

DCC - Merge Buffer (was DMA Cache)

MDU - Mondo Dispatch Unit

MMU - IOMMU

SBM - SBus Module

STC - Streaming Cache

Example of Signal Names

U2S_PA - UPA to SBus PIO Address bus

U2S_PD - UPA to SBus PIO Data bus

U2S_DA - UPA to SBus DMA Address bus

U2S_DD - UPA to SBus DMA Data bus

15.2 Overview

The “Mondo” interrupt transfer mechanism used for the UltraSPARC system reduces interrupt service overhead through the use of processor and system-based supports. On the processor side, UltraSPARC - 1 will provide a dedicated set of registers used exclusively for servicing interrupts. This eliminates the need for UltraSPARC - 1 to save its current register set to service an interrupt and then restore it later.

On the system side, requests for interrupt service are converted into interrupt request packets that are sent over the memory interconnect to the processor. An interrupt packet contains a Mondo vector which has three double-words designed to assist the processor in servicing the interrupt.

The Mondo vector approach is limited in the following ways:

- There is only one interrupt request packet that can be serviced at a time
- There is no priority level associated with Mondo vector interrupts; they are serviced on first come, first serve basis
- Flow control must be done at the interconnect level to prevent loss of interrupt packets

15.2.1 Mondo Dispatch Overview

The Mondo Dispatch Unit is responsible for fielding interrupts from external SBus sources and internal U2S sources, building the UPA Interrupt packet, and sending the interrupt packet to the appropriate CPU.

External interrupt sources include four SBus slots, the on-board IO devices, a Graphics Interrupt, and the expansion UPA slot. These interrupts are concentrated in an external ASIC and present to the Mondo Unit one at a time via the interrupt concentrator in the RIC chip. Internal interrupt sources include the ECC (errors), SBus Module (late errors), Timer Counters, and the Power Management Wakeup interrupt. These sources are discussed in further detail later.

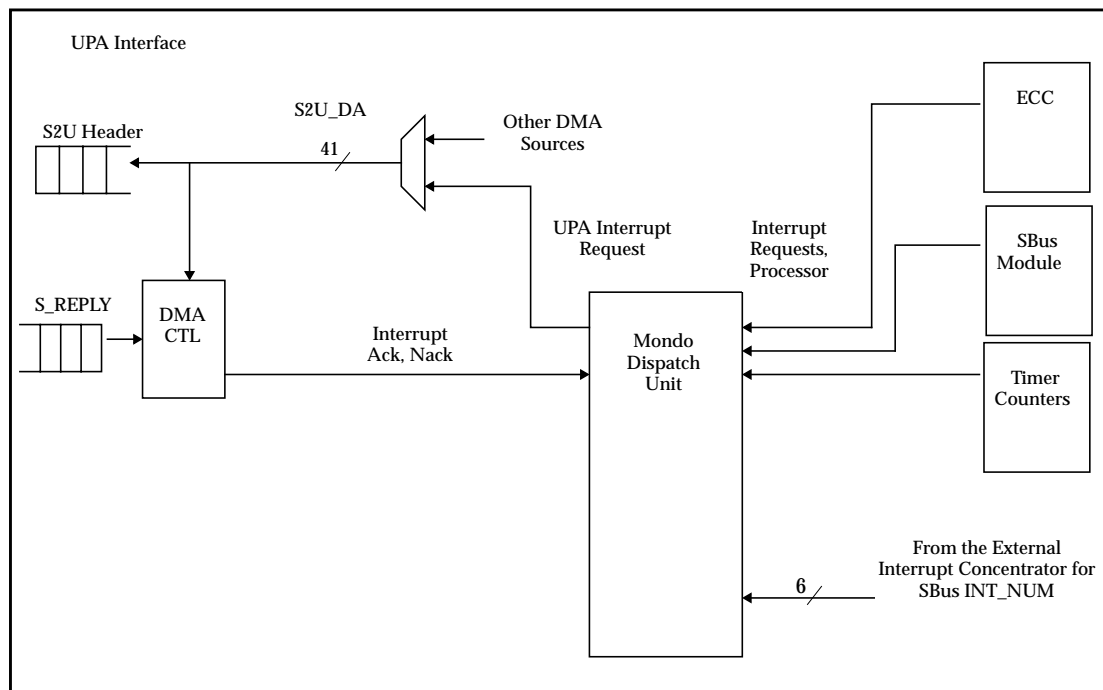


Figure 15-1 Mondo Dispatch Unit in U2S

15.2.2 Mondo Dispatch Block Diagram

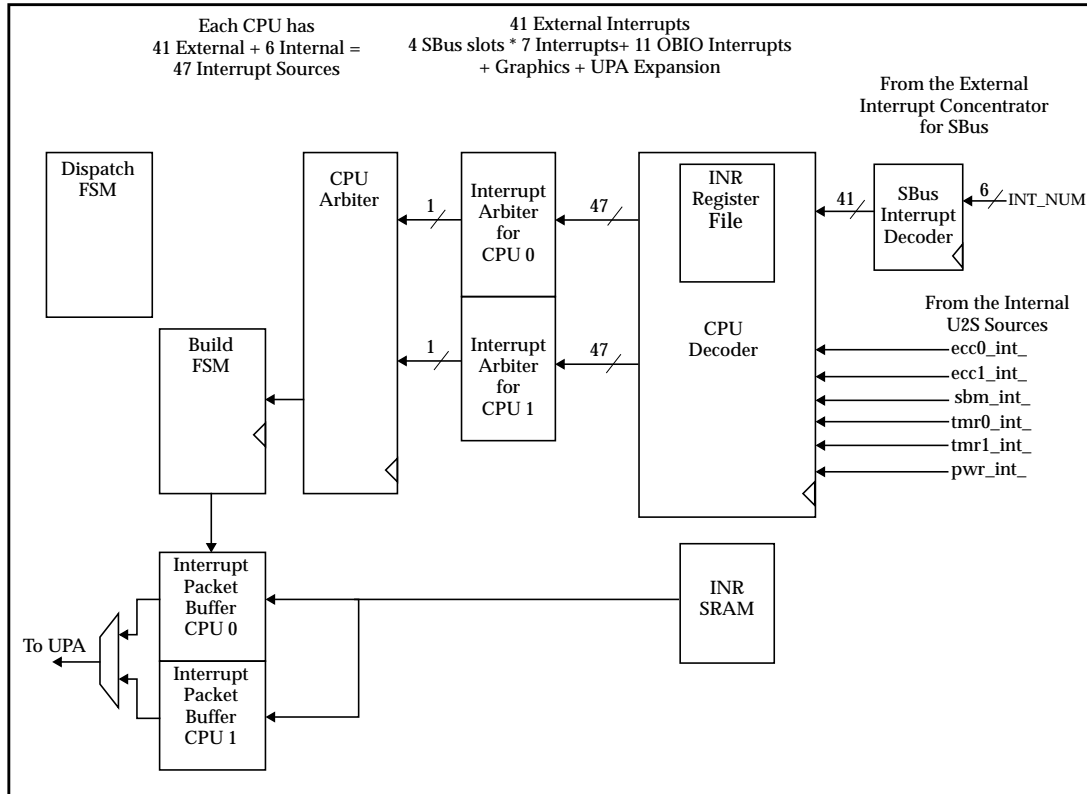


Figure 15-2 Mondo Dispatch Overview Block Diagram

Figure 15-2, “Mondo Dispatch Overview Block Diagram,” shows the general flow through the Mondo Dispatch Unit. Each triangle in a block indicates a clock cycle of latency. Thus, the overall latency is four cycles through the Mondo Dispatch Unit (from the External Interrupt Concentrator to issuing the request, to the internal U2S Bus Controller).

For more details, see Section 15.3, “Mondo Unit Functional Description.”

15.3 Mondo Unit Functional Description

The Mondo Unit is responsible for generating a UPA Mondo Vector Request Packet for interrupt clients. This section contains the following:

- An overview of Mondo Interrupts
- Interrupt Types
- Flow of an interrupt through the Mondo Dispatch Unit

15.3.1 Mondo Vectors

Before a functional discussion on the Mondo Dispatch Unit, it is necessary to provide a brief overview of Mondo Vectors. See the Sun-5 Architecture specifications for a more detailed description.

15.3.1.1 Overview of an Interrupt

Interrupts are delivered to the process in a packet format which looks like a 64 byte write on the UPA; this implies four cycles of 128 bits of data (or 8 cycles of 64 bits of data). However, only three double-words (3x32 bits) are used to carry “pertinent” information. Note that U2S does not deliver interrupt data, only the Interrupt Number.

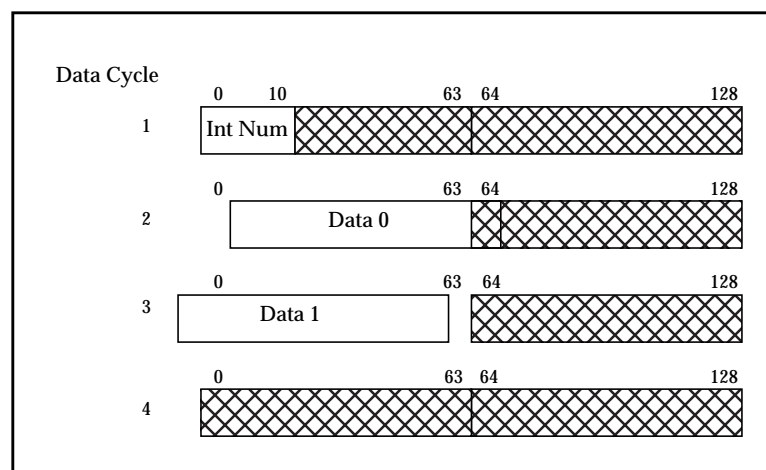


Figure 15-3 Mondo Vector Format on UPA Data Bus

The first data cycle contains the interrupt number (11 bits). The second and third data cycles contain 64 bits of data. These data fields contain interrupt specific information such as address, timer values, error information, status register values, etc. Again note that U2S does not deliver Data 0 and Data 1. All other bits are not used (driven to 0 on the data bus).

The interrupt number is specific to each interrupt source, which allows software to uniquely identify the source of the interrupt without having to poll all interrupt sources, thus reducing overhead in processing interrupts.

Note that there is no priority associated with the interrupt packet. Thus, interrupts are processed on a first-come-first-serve basis.

Each CPU can process only one interrupt at a time. All subsequent interrupt that are delivered to a busy CPU will get Naked on the UPA. The interrupt source must then retry later.

15.3.1.2 Interrupt Number Register

Generally, each interrupt source has an Interrupt Number Register (INR) associated with it. The INR is either fully or partially software programmable and contains the Interrupt Number, which is delivered in the first data cycle, the MID of the processor the interrupt is to be sent, and a valid bit which enables or disables the interrupt.

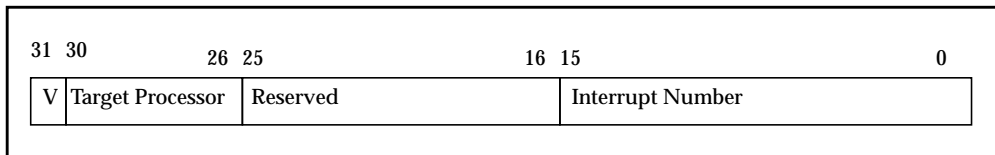


Figure 15-4 Full INR Contents

As shown the INR has 3 fields:

1. Valid bit (1 bit) - enables the interrupt when set to 1. Note that when an interrupt is present and the valid bit is 0, the interrupt is prevented from being delivered. However, once the valid bit is set to 1, the interrupt is delivered.
2. Target Processor (5 bits) - used to determine the address of the Interrupt in the UPA header. The Mondo Dispatch Unit also uses the LSB for arbitration.
3. Interrupt Number (16 bits) - delivered in the first data cycle.

For most of the interrupts, the Interrupt Number field is broken further broken down into two separate fields: the Interrupt Number Index and the Interrupt Number Offset. The Interrupt Number Offset is a fixed value depending on the interrupt.

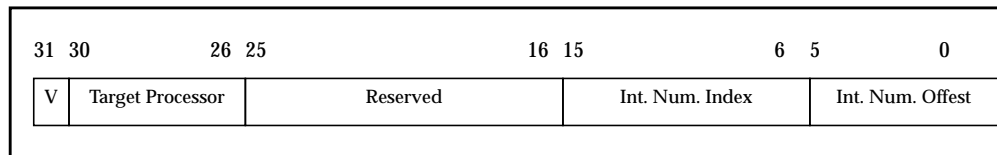


Figure 15-5 Partial INR Contents

For more information, see the UltraSPARC System Specifications.

15.3.2 Interrupt Types

Interrupts processed by the U2S are classified using three characteristics:

1. Internal/External.
2. Level/Pulse.
3. Priority.

15.3.2.1 Internal/External

Internal Interrupts

Internal Interrupts refer to those interrupts that are generated within U2S. Each internal interrupt source has a dedicated set of signals to the Mondo Unit for raising an interrupt. There are a total of 6 internal interrupts.

- ECC - The ECC unit will raise an interrupt when it detects a correctable or uncorrectable error for PIO Writes Requests and DMA Read Replies. There are 2 ECC interrupt lines, one for correctable and one for uncorrectable errors
- Power Management - The timer associated for Power Management will raise an interrupt to wakeup the U2S
- SBus Module - The SBus Module will raise an interrupt when there is a late error on the SBus for PIO Reads or Writes
- Timer Counters - The Timer Counters will raise an interrupt when its timer has elapsed. It will provide the limit of the counter as the data. There are 2 Timer Counter interrupts (1 for each Timer Counter)

External Interrupts

External Interrupts refer to those interrupts that are generated externally of U2S. All external sources for interrupts (SBus, OBIO, Graphics, and UPA) go through the Interrupt Concentrator. The Interrupt Concentrator logic resides in the RIC ASIC.

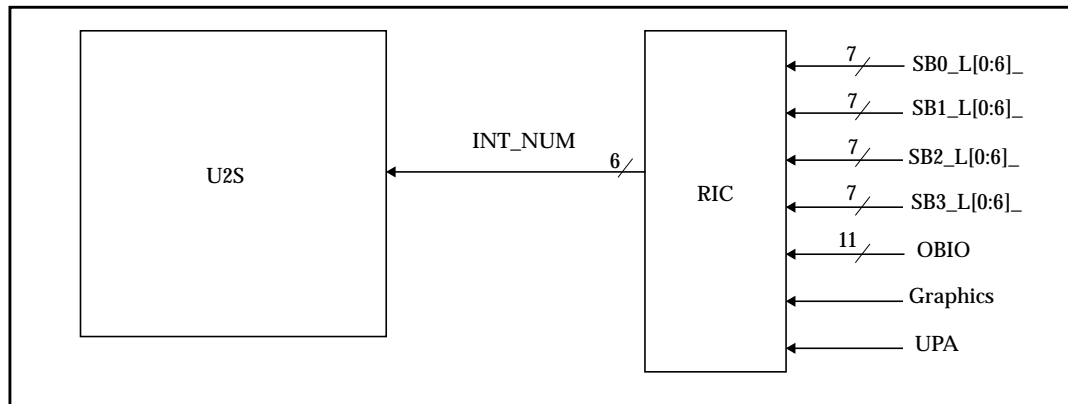


Figure 15-6 Interrupt Concentrator

The Interrupt Concentrator simply samples all interrupts lines and in round-robin fashion, presents one of them at a time to the U2S. The 41 interrupt lines are simply encoded into a 6 bit value to U2S. This was done to save pins on the U2S.

- SBus - U2S supports 4 SBus slots. Each SBus slot has 7 interrupt lines (levels). So, there are 28 interrupt lines from SBus
- OBIO (On-board IO Devices) - There are 11 OBIO devices
- Graphics/UPA - 2 UPA slot interrupts are supported. These are the only two interrupts that are of pulse type (see below). These are also the only interrupts with the full INR register (fully software programmable). All other interrupts have a INX and INO fields

15.3.2.2 Level/Pulse

An interrupt can be either level or pulse driven. Level interrupts have three states associated with them: Idle, Received, and Pending (shown below). Level interrupts include all interrupts except for the Graphics and UPA interrupts.

Table 15-1 Level Interrupt States

| State | Description |
|-------|--|
| 00 | Idle - no interrupt has been received yet |
| 01 | Received - the source has raised an interrupt; the interrupt is going through decoding and arbitration |
| 11 | Pending - the interrupt has won arbitration; it will be or has been delivered; the interrupt is disabled until a process PIO writes the state register to set the interrupt back to IDLE |
| 10 | Reserved |

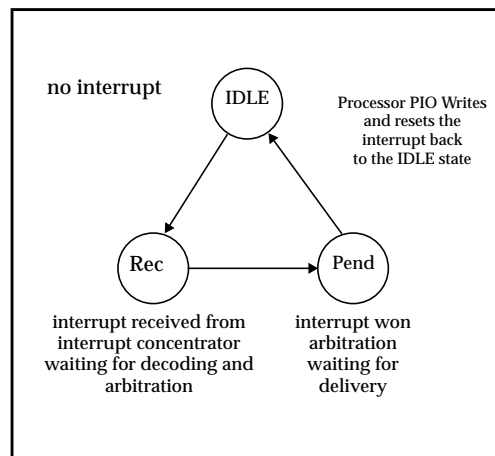


Figure 15-7 Level Interrupt States

Pulse Interrupts only have two states: Idle and Received. After the U2S delivers the interrupt, the interrupt returns to the Idle state. No software intervention is needed as in the Level interrupts case. Pulse interrupts include the Graphics and UPA Interrupts.

15.3.2.3 Priority

Each interrupt has a priority associated with it. There are a total of 8 priority levels (8 is the highest priority, 1 is the lowest).

Priority is taken into account during interrupt arbitration. When multiple interrupts are present, the highest priority interrupt is delivered first. If multiple interrupt with the same priority are present, the interrupts are delivered in a round-robin fashion. When all interrupts at the highest priority level are delivered, the next highest priority level is processed.

Table 15-2 Interrupt Receiver State Register

| Level | Number of Interrupts | Source |
|-------|----------------------|---|
| 8 | 6 | Audio, Power Fail, Floppy, UE ECC, CE ECC, SBus Async Error |
| 7 | 6 | 4 SBus Slots, Kbd/mouse/serial, Serial Int |
| 6 | 6 | 4 SBus Slots, Timer 0, Timer 1 |
| 5 | 6 | 4 SBus Slots, OB Graphics, UPA64S Int |
| 4 | 6 | 4 SBus Slots, Keyboard Int, Mouse Int |
| 3 | 6 | 4 SBus Slots, SCSI Int, Ethernet Int |
| 2 | 6 | 4 SBus Slots, Parallel Port, Spare Int |
| 1 | 5 | 4 SBus Slots, Power Management |

Table 15-3 Summary of Interrupts

| Interrupt | Int/Ext | Source | INT_NUM code | Type | Offset | Priority |
|----------------------|---------|--------|--------------|-------|--------|----------|
| SBus Slot 0, Level 1 | Ext | SBus | 000001 | Level | 000001 | 1 |
| SBus Slot 0, Level 2 | Ext | SBus | 000010 | Level | 000010 | 2 |
| SBus Slot 0, Level 3 | Ext | SBus | 000110 | Level | 000110 | 3 |
| SBus Slot 0, Level 4 | Ext | SBus | 000100 | Level | 000100 | 4 |
| SBus Slot 0, Level 5 | Ext | SBus | 000101 | Level | 000101 | 5 |
| SBus Slot 0, Level 6 | Ext | SBus | 000110 | Level | 000110 | 6 |
| SBus Slot 0, Level 7 | Ext | SBus | 000111 | Level | 000111 | 7 |
| SBus Slot 1, Level 1 | Ext | SBus | 001001 | Level | 001001 | 1 |
| SBus Slot 1, Level 2 | Ext | SBus | 001010 | Level | 001010 | 2 |
| SBus Slot 1, Level 3 | Ext | SBus | 001110 | Level | 001110 | 3 |
| SBus Slot 1, Level 4 | Ext | SBus | 001100 | Level | 001100 | 4 |
| SBus Slot 1, Level 5 | Ext | SBus | 001101 | Level | 001101 | 5 |
| SBus Slot 1, Level 6 | Ext | SBus | 001110 | Level | 001110 | 6 |
| SBus Slot 1, Level 7 | Ext | SBus | 001111 | Level | 001111 | 7 |
| SBus Slot 2, Level 1 | Ext | SBus | 010001 | Level | 010001 | 1 |
| SBus Slot 2, Level 2 | Ext | SBus | 010010 | Level | 010010 | 2 |
| SBus Slot 2, Level 3 | Ext | SBus | 010110 | Level | 010110 | 3 |
| SBus Slot 2, Level 4 | Ext | SBus | 010100 | Level | 010100 | 4 |
| SBus Slot 2, Level 5 | Ext | SBus | 010101 | Level | 010101 | 5 |
| SBus Slot 2, Level 6 | Ext | SBus | 010110 | Level | 010110 | 6 |
| SBus Slot 2, Level 7 | Ext | SBus | 010111 | Level | 010111 | 7 |
| SBus Slot 2, Level 1 | Ext | SBus | 011001 | Level | 011001 | 1 |
| SBus Slot 3, Level 2 | Ext | SBus | 011010 | Level | 011010 | 2 |
| SBus Slot 3, Level 3 | Ext | SBus | 011110 | Level | 011110 | 3 |
| SBus Slot 3, Level 4 | Ext | SBus | 011100 | Level | 011100 | 4 |

Table 15-3 Summary of Interrupts

| Interrupt | Int/Ext | Source | INT_NUM code | Type | Offset | Priority |
|----------------------|---------|--------|--------------|-------|--------|----------|
| SBus Slot 3, Level 5 | Ext | SBus | 011101 | Level | 011101 | 5 |
| SBus Slot 3, Level 6 | Ext | SBus | 011110 | Level | 011110 | 6 |
| SBus Slot 3, Level 7 | Ext | SBus | 011111 | Level | 011111 | 7 |
| SCSI | Ext | OBIO | 100000 | Level | 100000 | 3 |
| Ethernet | Ext | OBIO | 100001 | Level | 100001 | 3 |
| Parallel Port | Ext | OBIO | 100010 | Level | 100010 | 2 |
| Audio | Ext | OBIO | 100100 | Level | 100100 | 8 |
| Power Fail | Ext | OBIO | 100101 | Level | 100101 | 8 |
| Kbd/Mouse/Serial | Ext | OBIO | 101000 | Level | 101000 | 7 |
| Floppy | Ext | OBIO | 101001 | Level | 101001 | 8 |
| Spare Hardware | Ext | OBIO | 101010 | Level | 101010 | 2 |
| Keyboard | Ext | OBIO | 101101 | Level | 101101 | 4 |
| Mouse | Ext | OBIO | 101110 | Level | 101110 | 4 |
| Serial | Ext | OBIO | 101111 | Level | 101111 | 7 |
| Timer 0 | Int | Timers | | Level | 110000 | 6 |
| Timer 1 | Int | Timers | | Level | 110001 | 6 |
| Uncorrectable ECC | Int | ECC | | Level | 110100 | 8 |
| Correctable ECC | Int | ECC | | Level | 110101 | 8 |
| SBus Error | Int | SBM | | Level | 110110 | 8 |
| Power Management | Int | Timers | | Level | 110111 | 1 |
| Graphics | Ext | UPA | 100011 | Pulse | | 5 |
| UPA Expansion Slot | Ext | UPA | 100110 | Pulse | | 5 |

15.3.3 Processing an Interrupt

The Mondo Dispatch Unit in U2S is optimized for a two processor system. The Mondo Dispatch Unit is composed of four main blocks: External Interrupt Receiver, Interrupt Decoder, Arbiter, and the Dispatcher. These blocks are described in the following sections:

15.3.3.1 External Interrupt Receiver

The External Interrupt Receiver receives interrupt numbers from the Interrupt Concentrator one at a time. It decodes the interrupt and stores it in a 2 bit state register. There is 1 state register for each interrupt source (47 total). The Graphics and UPA interrupts have a 1 bit state register, since they are Pulse Interrupts.

When an interrupt is in the received state, its interrupt line will be asserted to the next block, the Interrupt Decoder block.

15.3.3.2 Interrupt Decoder

The Interrupt Decoder uses the INR for each interrupt to determine the CPU that interrupt is to be delivered. 41 interrupt lines from the External Interrupt Receiver is fed into this block, as well as the 6 internal interrupt sources (2 ECCs, Power Management, 2 Timer Counters, and SBus Module). The Mondo Unit keeps the INR of all interrupts, external and internal.

The output of the Interrupt Decoder is 2 sets of 47 interrupt lines (47 interrupt lines for each CPU). This is fed into the next block, the Interrupt Arbiter.

15.3.3.3 Interrupt Arbiter

The Interrupt Arbiter arbitrates among two sets of 47 interrupt lines and chooses a winner. The first stage of arbitration involves choosing one winner for each CPU. The highest priority level interrupts are chosen first. Then a round-robin among those interrupts picks the winner.

After a winner has been chosen for each CPU, a round-robin chooses between the two CPUs for a winner. This is fed into the next block, the Interrupt Dispatcher.

15.3.3.4 Interrupt Dispatcher

The Interrupt Dispatcher is composed of the following three sub-blocks:

1. Packet Builder FSM.

The Packet Builder takes the winner from the Interrupt Arbiter and assembles the interrupt packet in the appropriate CPUs Packet Buffer. There is 1 Packet Buffer for each CPU. Once the Packet Buffer contains an interrupt, it prevents that CPU from winning arbitration until the buffer is cleared (for example: the interrupt delivered and Acked).

The Packet Builder reads the INR valid from the local SRAM and places it in the Packet Buffer.

2. Dispatcher FSM.

The Dispatcher checks each Packet Buffer in a round-robin fashion. If the Packet Buffer contains a valid interrupt that is ready to be sent, the Dispatcher will raise a request to the U2S Bus Controller to deliver an interrupt packet (which looks like a 64 byte write) to the UPA interface.

After delivering the interrupt, the Dispatcher waits for the Ack or Nack from the System Controller. If the interrupt is Acked, the Packet Buffer is cleared. If the interrupt is Nacked, the Dispatcher clears the Retry Bit in the Packet Buffer. In both cases, the Dispatch proceeds to the next Packet Buffer.

Note: Note that this means that the Mondo Unit dispatches only 1 interrupt at a time, and waits for the Ack or Nack before dispatching the next interrupt.

3. Retry FSM.

There is a Retry FSM associated with each Packet Buffer. When the Retry Bit is cleared (by the Dispatcher), the Retry FSM waits for a common free-running counter to roll over twice then sets the Retry Bit.

Setting the Retry Bit sets the Packet Buffer into the ready state for when the Dispatcher comes around the next time.

15.4 Mondo Dispatch Timing Diagrams

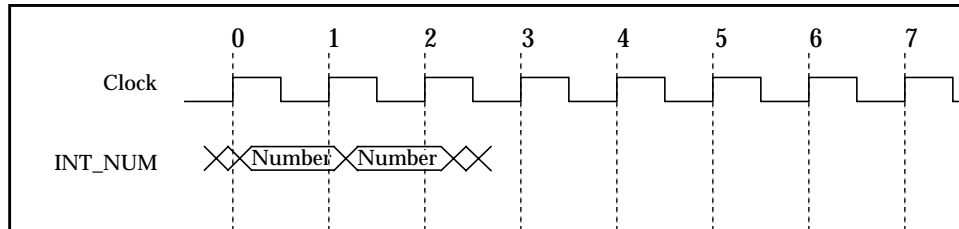


Figure 15-8 External Interrupt Concentrator Timing

Each cycle, the External Interrupt Concentrator may present a different Interrupt Number to U2S. The following table lists the external interrupt encoding.

Table 15-4 External Interrupt Encoding

| Encoding | Interrupt |
|----------|--------------------------|
| 000_001 | SBus Slot 0, Interrupt 1 |
| 000_010 | SBus Slot 0, Interrupt 2 |
| 000_011 | SBus Slot 0, Interrupt 3 |
| 000_100 | SBus Slot 0, Interrupt 4 |
| 000_101 | SBus Slot 0, Interrupt 5 |
| 000_110 | SBus Slot 0, Interrupt 6 |
| 000_111 | SBus Slot 0, Interrupt 7 |
| 001_001 | SBus Slot 1, Interrupt 1 |
| 001_010 | SBus Slot 1, Interrupt 2 |
| 001_011 | SBus Slot 1, Interrupt 3 |
| 001_100 | SBus Slot 1, Interrupt 4 |
| 001_101 | SBus Slot 1, Interrupt 5 |
| 001_110 | SBus Slot 1, Interrupt 6 |
| 001_111 | SBus Slot 1, Interrupt 7 |
| 010_001 | SBus Slot 2, Interrupt 1 |
| 010_010 | SBus Slot 2, Interrupt 2 |

Table 15-4 External Interrupt Encoding

| | |
|---------|--------------------------|
| 010_011 | SBus Slot 2, Interrupt 3 |
| 010_100 | SBus Slot 2, Interrupt 4 |
| 010_101 | SBus Slot 2, Interrupt 5 |
| 010_110 | SBus Slot 2, Interrupt 6 |
| 010_111 | SBus Slot 2, Interrupt 7 |
| 011_001 | SBus Slot 3, Interrupt 1 |
| 011_010 | SBus Slot 3, Interrupt 2 |
| 011_011 | SBus Slot 3, Interrupt 3 |
| 011_100 | SBus Slot 3, Interrupt 4 |
| 011_101 | SBus Slot 3, Interrupt 5 |
| 011_110 | SBus Slot 3, Interrupt 6 |
| 011_111 | SBus Slot 3, Interrupt 7 |
| 100_000 | SCSI |
| 100_001 | Ethernet |
| 100_010 | Parallel Port |
| 100_011 | Graphics |
| 100_100 | Audio |
| 100_101 | Power Fail |
| 100_110 | UPA |
| 101_000 | Keyboard/Mouse/Serial |
| 101_001 | Floppy |
| 101_010 | Spare Int |
| 101_101 | Keyboard |
| 101_110 | Mouse |
| 101_111 | Serial |
| 111_111 | None |

There are two independent timers in the U2S. Each provides either periodic interrupts or alarm-clock (callout) interrupts to a selected processor. One of the timers has a prescaler to provide walk-up interrupts when the system is in the *SLEEP* mode.

16.1 Overview

The features supported are as follows:

- Limit Register is used to set the interrupt enable, reload, periodic bits, and the counter interrupt comparison value
- Count Register is used for loading the counter on write, and used to return the current count on read
- Periodic interrupts can be generated
- Interrupt can be disabled
- 29-bit counter will allow a maximum count of 0x1FFFFFFF, or 536 seconds using a one microsecond count interval. During power management mode, one of the time counters will use a one millisecond count interval and will provide the walk-up interrupt when the count equals the limit register value

16.1.1 Timer/Counter Overview

The major blocks in the Timer unit are:

- 29-bit up counter with synchronous load. For testability, this counter is decomposed into six 4-bit counters and one 5-bit counter
- Limit Register for setting time-out value and other control bits
- Count Register for loading the timer with a preset value, and for reading the current value of counter
- Comparator to compare counter output with the limit register
- Mondo Vector Generation Unit for generating interrupt request
- Inbound and outbound synchronization flip flops
- Output Datapath for driving diagnostics read data

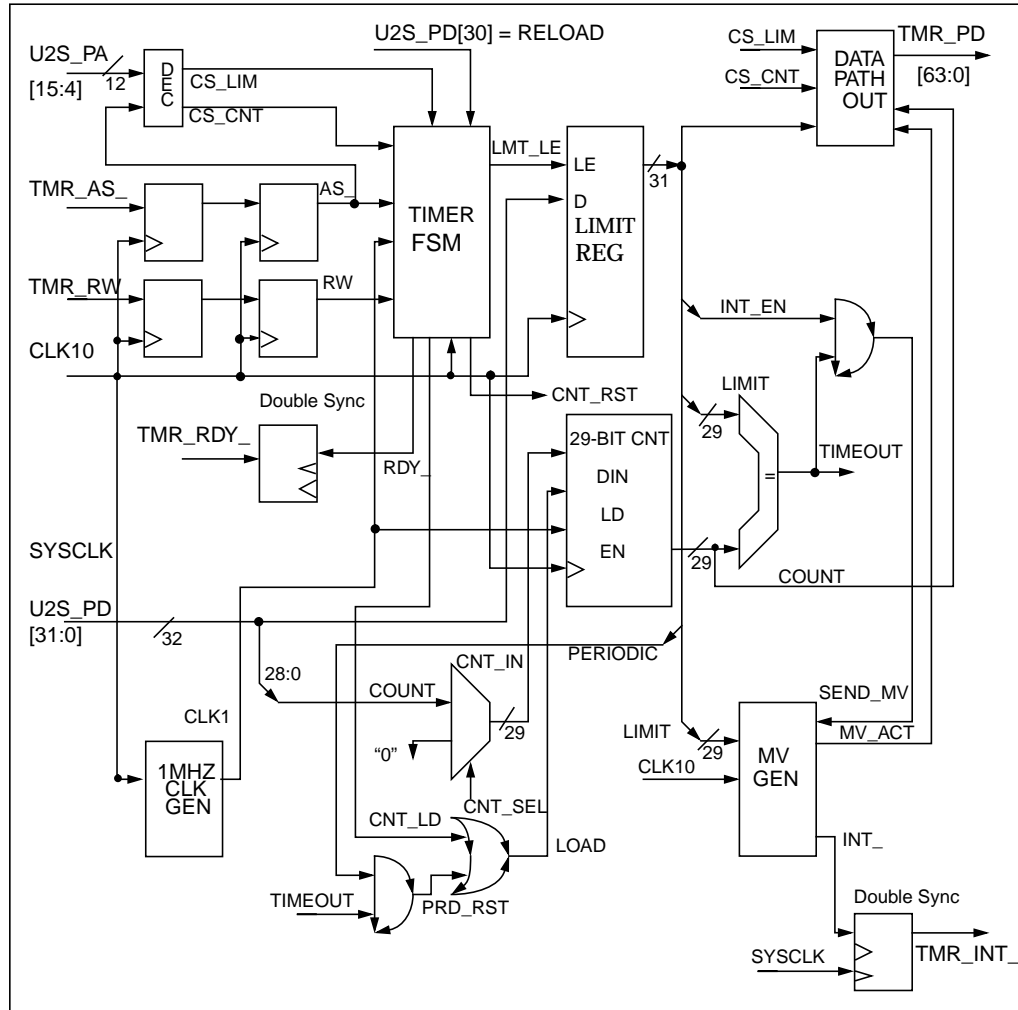


Figure 16-1 Timer Block Diagram

16.2 Timer Signal Descriptions

Table 16-1 Timer Signals

| Signal Name | Signals | I/O | Description |
|--------------|---------|-----|--|
| U2S_PA[13:3] | 11 | I | Internal UPA address for PIO access |
| TMR_AS_ | 1 | I | Address Strobe indicating address valid during PIO access |
| TMR_RW | 1 | I | PIO request. 1 means PIO read, 0 means PIO write |
| CLK10 | 1 | I | 10 MHZ clock from Ethernet oscillator |
| SIOCLK | 1 | I | U2S clock |
| SIORST_ | 1 | I | Reset with respect to U2S clock |
| TMRRST_ | 1 | I | Reset with respect to CLK10 clock |
| U2S_PD[63:0] | 32 | I | Internal UPA data bus for diagnostics /PIO access. Note bits [63:32] are not used since timer registers are always accessed using 32-bit PIO |
| TMR_PD[63:0] | 64 | O | Output from Timer internal registers |
| TMR_RDY_ | 1 | O | Indicate Timer has driven valid data in response to PIO read, or that Timer has taken the data in response to PIO Write |
| TMR0_INT_ | 1 | O | Interrupt request from Timer 0. Asserted during time-out |
| TMR1_INT_ | 1 | O | Interrupt request from Timer 1. Asserted during time-out |
| TWU_INT_ | 1 | O | Timer walk-up interrupt |
| PWR_CTRL_EN | 1 | I | Power Management Control bit |

16.3 Timer Functional Descriptions

It is expected that the two timers will have independent functions; one used for system collate events, and the other for operating system profiling. It is up to the processor to issue cross-calls to other processors if broadcast is needed. Each timer has separate Count and Limit registers. In order for the processor to uniquely identify the source of the timer interrupts, each timer will also have its own Interrupt Number Register.

The two timers operate as follows:

- Writes to the Limit Register set the LIMIT value, and causes the corresponding timer to reset to zero if (RELOAD == 1)
- Else if (RELOAD != 0), then the LIMIT gets set without affecting the value of COUNT
- When (COUNT == LIMIT) and (INT_EN == 1), an interrupt request is made. When granted, the Mondo Vector with the corresponding INR is dispatched
- If (PERIODIC == 1), whenever (COUNT == LIMIT) then the counter is reset to zero and continues counting
- Else if (PERIODIC == 0), when (COUNT == LIMIT) then the counter continues to count normally without taking an intermediate reset. Note that the counter will still wrap-around to zero if the current count has reached 0xFFFFFFFF
- To obtain a periodic interrupt every 'N' microseconds, the LIMIT should be set to 'N-1'
- The walk-up enable (also called the pwr_ctrl_en) bit enables a divide by 1000 prescaler for one of the timer counters thereby changing them from incrementing once per microsecond to once per millisecond

Note: If (INT_EN == 0), when (COUNT == LIMIT) then the counter will not send interrupt. However, counter might be reset depending on the state of PERIODIC.

16.3.1 Timer FSM Descriptions

16.3.1.1 Overview

The Timer FSM provides the control signals for the PIO access to both timers.

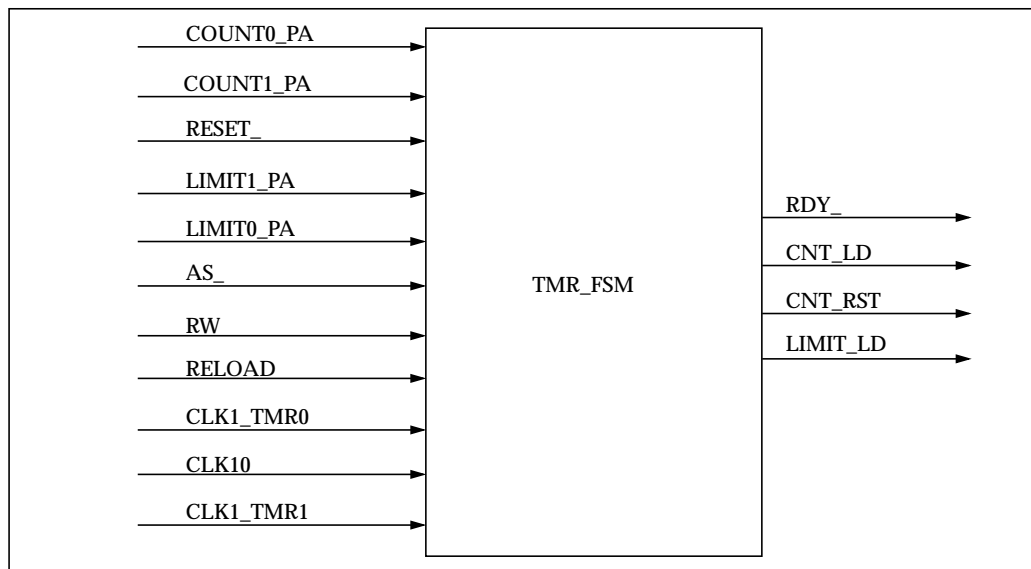


Figure 16-2 TMR_FSM Block Diagram

16.3.1.2 TMR_FSM State Transition Diagram

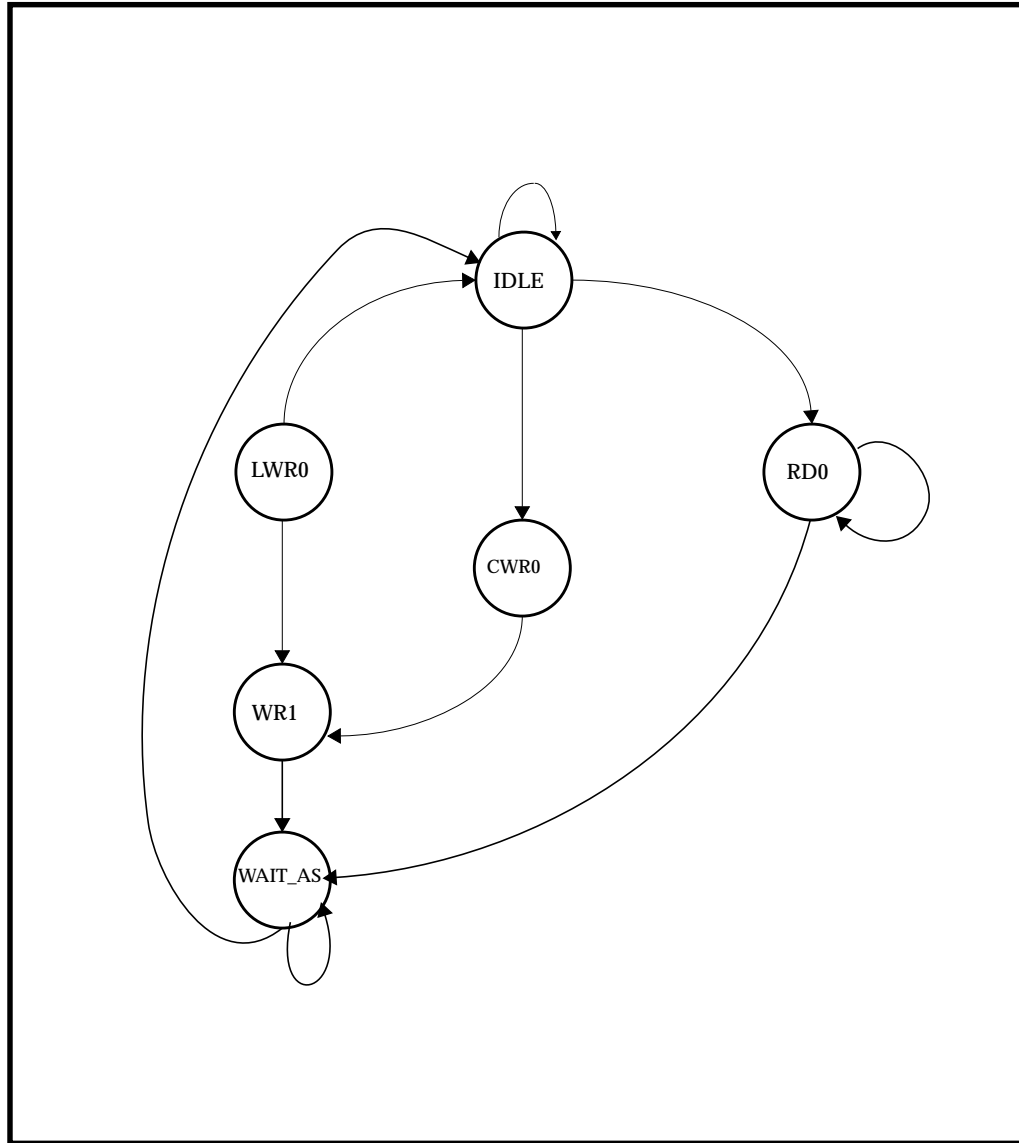


Figure 16-3 TMR_FSM State Diagram

16.3.2 29-Bit Counter Descriptions

Each timer will have a 29-bit counter constructed from a cascade of six 4-bit counters and one 5-bit counter.

Each counter will have on its input a LOAD pin (for synchronous reset and loading non-zero values), a COUNTER ENABLE pin, 4-bit input data (5-bit for the MSB counter), CLOCK pin, and CARRY-IN. The output consists of 4-bit output data and CARRY-OUT to the next counter.

16.3.3 1 MHz Clock Generator Descriptions

This unit does a divide by 10 of the 10-MHZ Ethernet oscillator input to produce a 1-MHZ clock. Each timer has this unit. The timer which provides the walk-up enable interrupt also has the additional prescaler logic.

16.3.4 Output Data Path Description

This unit is responsible for driving the proper data in response to PIO read on the Limit/Count Registers.

16.3.5 Mondo Vector Unit Descriptions

16.3.5.1 Overview

The main function of this block is to request and send Interrupt at the occurrence of time-out.

This block functions independently of the normal counting. When the processor is not available to handle the interrupt request, this unit simply waits until its request is granted.

Note: If the current interrupt can not be dispatched, all subsequent time-out interrupts will be ignored.

17.1 Overview

This section describes the error detection and correction, as well as the supported error reporting mechanisms. Errors related to an UltraSPARC-I processor are described in the *UltraSPARC-I Programmer's Reference Manual*.

Errors detected in the system are classified into the following types: fatal errors and non-fatal hardware errors. A fatal error may result in a system reset if the error reset is enabled by software. Actions taken on non-fatal hardware errors include interrupts, status registers, or none.

17.2 Error Detection and Reporting

The reference platform provides error detection at interconnect levels and inside each U2S. Parity and ECC are used to protect the interconnects at various levels.

17.2.1 Error Description

17.2.1.1 UPA Address Parity Error

The UPA address bus carries address and control information for the UPA transactions. The format of an address packet is described in the UPA Interconnect Architecture specification. One address bus connects between processor(s), the U2S and the USC. Only the address bus connecting processor(s), the U2S and the USC, support(s) parity. The parity bit is driven by a master UPA port at the same time the address packet is driven to the bus. The parity bit will be set to 1 when there is an even number of 1s on the address bus. The USC receiving the address

packet will perform parity checking. If the address packet targeted to another UPA slave port has no parity error, the USC will route the address packet to the target device on the other address bus.

An address parity error is considered a fatal error to the system. Both the UPA slave port and the System Controller USC can detect address parity errors. The actions taken on the parity error are described below:

- *Address parity error detected by a UPA slave port:* The device detecting an address parity error should send a P_FERR reply to the USC if it is enabled to do so. The USC will generate reset to the system upon receipt of P_FERR. The USC will log the fatal error condition in the USC Port Status Register and the USC Control Register
- *Address parity error detected by the System Controller:* The USC generates reset to the system and logs an error condition in the USC Control Register

Software should clear the error bit by writing one to it. Failing to do so may cause confusion to the software if another address parity error happens after the E-Cache data parity error.

17.2.1.2 UPA Datapath Uncorrectable Error

The UPA datapath implemented in the reference platform consists of the Cross-bar Switch (XB1), and a collection of wires connecting the UltraSPARC-I Data Buffer (UDB), XB1, memory, U2S, and UPA graphic devices. There are three major busses provided for this purpose:

- *Memory Data Bus*, 256 bits of data and 32 bits of ECC information
- *UPA Data Bus (Processor)*, 128 bits of data and 16 bits of ECC information
- *UPA Data Bus (I/O)*, 64 bits of data and 8 bits of ECC information

Devices connected to the UPA datapath perform ECC generation, checking, and correction. In the reference platform, devices supporting ECC generation, checking, and correction are the UDB and the U2S.

Two types of ECC errors are checked by the UPA port, Correctable Error (CE) and Uncorrectable Error (UE). CE can come from the following sources:

- *Corruption on the datapath*, such as UPA, and memory
- *Faulty device*, such as DRAM, XB1, or UPA device

In addition to normal UE sources, there are other cases a device can force a UE to the system. The UE handling software needs to find out what exactly caused the UE in the system. Information relating to UltraSPARC-I-induced UE is described in UltraSPARC-I AFSR/AFAR. Information relating to the U2S-induced UE is described in the U2S SBus Control Register section. Possible sources of the UE are listed below:

- *Corruption on the datapath*, such as UPA, and memory
- *Faulty device*, such as DRAM, XB1, or UPA device
- E-cache data parity error
- SBus data parity error during DVMA write
- SBus data parity error during PIO read

If the device detects a CE, data will be corrected before it is used. Data transfer continues as if there is no error. The receiving device will log the error information and report the error. Upon an UE, the error will be carried over, maybe in a different way, to make sure the erroneous data is not consumed. The receiving device will log the error information and report the error.

The U2S detects and corrects ECC errors on the data it receives. The checking and correction are done on the following operations:

- The PIO writes to the U2S and devices controlled by the U2S
- The SBus DVMA read from memory or the UPA devices
- The SBus DVMA write to memory

The U2S reports an ECC error to the processor by interrupt if ECC checking and ECC interrupt are both enabled. Error information is logged in the UE and CE AFSR/AFAR. For more information about the ECC control register, UE, and CE error registers please read the programming model section.

The other device that performs ECC checking/correction is the UltraSPARC-I UDB. The UDB checks/corrects ECC errors on processor instruction/data fetch and interrupt packets. Actions taken by the UltraSPARC-I on ECC errors are described in the *UltraSPARC-I Programmer's Reference Manual*.

17.2.1.3 UPA Timeout

UPA timeout can happen during a slave read to a non-existing UPA port or access to an unsupported/nonexisting device controlled by the UPA port. For example, reading from an empty SBus slot.

A UPA transaction needs to be completed with a reply in order to avoid hanging the initiating device. There is no bus timer defined in the UPA interconnect to keep track of pending transactions. To avoid hanging the system due to an inadvertent access to a non-existing device, the USC will check for the existence of the device before relaying the access to the targeted port. A UPA port should signal its existence by sending an IDLE reply to the USC when there is no access to it. If the port exists in the system, it is the responsibility of the UPA port to issue a reply to any access targeted to it.

Access to a non-existing UPA port is handled differently for slave read and slave write.

- Slave write to a non-existing UPA port will be ignored by the SC. The IADD bit in the USC Port_Status Register will be set
- Slave read from a non-existing port will be terminated with a read timeout reply from the USC, and the IADD bit in the USC Port_Status Register will be set. The handling of a read timeout error is device dependent

A PIO read from a non-existing or faulty SBus device will also result in a UPA timeout error reported to the initiating UPA master.

17.2.1.4 UPA Read Error

Other than address parity error, the UPA does not provide an error reply to write transactions. The USC or UPA port can only send an error reply to read transactions. Write errors detected by the UPA slave port should be reported through an interrupt. A UPA slave can send an UPA read error reply for various reasons described below. For example, the U2S will respond with a read error reply if the SBus error acknowledge is received from the SBus slave device. The USC will return an error reply to the master port if an error reply is received from an UPA slave, or if an UPA master attempts to perform a non-cached UPA transaction to memory space. In the latter case, the IADD bit of the Port_Status register will be set to reflect the error.

17.2.1.5 SBus Parity Error

The SBus provides parity protection on its datapath. A single-bit parity is provided for 32-bit data in the normal transfer mode and 64-bit data in extended transfer mode. Odd parity is used for 32-bit/64-bit data transfers and data path parity transfer.

In order to enable parity checking, devices involved in the data transfer must support parity generation and checking capability. The FCode on the SBus device provides the parity attribute. System software can examine the FCode to find out whether a SBus device supports parity or not. The way to enable parity checking on the SBus device is device dependent. It is not described in this document.

SBus Slot Configuration Registers (SSCR) in the U2S control the parity checking and generation in the U2S during SBus data transfer. One SSCR is provided to each SBus device. Setting the parity enable (PE) bit of this register to 1 will enable the U2S to perform parity checking on transfers to/from the SBus device.

The U2S acting as the SBus controller/master/slave performs parity checking in the following situations if parity checking is enabled for the device.

- *Translation Phase* when the SBus master device drives virtual address to D[31:0] of the SBus. The U2S will return a SBus error acknowledge to the SBus device. It is up to the device to report the error
- *Extended Transfer Information (ETI) Phase* when the SBus master device provides extended transfer information to the U2S. The U2S will return a SBus error acknowledge to the SBus device. It is up to the device to report the error
- *Data Transfer Phase and Extended Data Transfer Phase* when the U2S is receiving data from the device. This can happen on a PIO read from the SBus device, or SBus DVMA write to the U2S/UPA/memory

A parity error detected on the PIO read will be reported with uncorrectable data being delivered to the processor and the PIO error bit in the SBus control register set. A parity error detected during a SBus DVMA write will be reported with a SBus late error to the SBus master with proper timing. Action taken by the master on the SBus late error is device dependent. If the DVMA write is targeted to the U2S, the write will be ignored. If it is targeted for the UPA device or memory, the U2S will provide data with uncorrectable error, and set the DMA_PERR status bits in the U2S SBus Control Register. Error handling software needs to clear the bits after servicing the UE.

An SBus device can also detect a parity error during the Data Transfer Phase or Extended Data Transfer Phase of the PIO write or DVMA read. The error reporting mechanism is device specific.

17.2.1.6 SBus Timeout

The U2S maintains a timer to keep track of slave accesses to SBus devices. If the device does not exist, or exists but does not respond to the access over a period of time, the U2S will terminate the cycle with SBus error acknowledge when the timer expires. The timer starts ticking when the SBus slave cycle starts with SBus Address Strobe. The timeout period is defined by the SBus specification to be 256 SBus clocks.

The SBus timeout on the PIO access is reported differently depending on the type of access. A PIO read terminated by the SBus timeout will be reported with the UPA timeout reply to the initiating UPA port. The SBus write timeout will be reported through the interrupt. Please refer to the programming model section for the logging of error information.

The SBus timeout can also happen while the SBus master directly accesses the SBus slave device. In this case, the handling of the SBus timeout is device dependent and is described in the SBus specification.

17.2.1.7 SBus Error

The SBus slave device can signal an error to the master access due to various error conditions. Examples of error conditions detected by the slave devices are unsupported address space, accesses with the wrong size, protection error, error conditions internal to the device, etc.

A SBus error happening during a PIO read will be reported with a UPA error reply to the processor. A SBus error happening during a PIO write will be reported as an interrupt and the U2S will log error information in the SBus AFSR/AFAR.

The U2S can also return an error acknowledge to the SBus master during DVMA transfers. The error can be caused by UE at the memory or UPA interconnect, translation error, access to non-existing UPA device, error internal to the UPA device, etc. The reporting of the SBus DVMA error is device dependent.

17.2.1.8 SBus Late Error

The SBus allows the slave device to report an error condition after the proper acknowledge is given to the master device. The timing of SBus late error assertion is defined in the SBus specification. A late error can happen during both PIO and DVMA accesses. A late error during a PIO read/write access will be reported by interrupts and error information will be logged in the SBus AFSR/AFAR.

The U2S will assert a late error if it detects a parity error during Extended Data Transfer Phases or the Data Transfer Phase of DVMA write transfers. Please refer to the SBus Parity Error section for the action taken by the U2S. The handling of the SBus DVMA late error is device specific.

17.2.1.9 DVMA Errors

The UPA UE registers log information during some DVMA errors. The following is a summary of the reporting:

1. If the UE interrupt is enabled, an interrupt will be posted when the U2S detects an UE.
2. Data will be ignored by the Streaming Buffer if the UE is caused by a Streaming Buffer prefetch. The entry will stay invalid if an UE is detected.

An UE caused by a demand DVMA read from a streamable page will cause an error acknowledge to the SBus device even if the UE does not happen on the 8-byte quantity requested by the device. This is to avoid paying one SBus cycle penalty on a read hit because the error status is not available when the SBus acknowledge is asserted. The Streaming Buffer entry will be set to an invalid state if this happens.

- An UE caused by a DVMA read from a consistent page will cause an error acknowledge to the SBus device only if the UE happens on the data requested. The SBus data acknowledge will be provided if the requested bytes do not have an error
- An UE on DVMA partial write: if a DVMA transaction totally writes over the eight bytes getting the UE, the error will be written over. Good data and check bits are provided for the eight bytes when writing it back to memory. If a DVMA transaction partially overwrites or does not overwrite the data having the UE, the U2S will force the UE to memory
- If a SBus parity error is detected during a DVMA write, the U2S will force the UE only to the bytes having a SBus parity error. If there is no SBus parity error within the aligned 8-byte data, good data and check bits will be provided to memory

If a SBus parity error is detected during a PIO read, the U2S will force the UE only to the bytes having a SBus parity error.

17.2.1.10 IOMMU Translation Error

During a SBus DVMA operation, the IOMMU will provide translation to the SBus virtual address. The IOMMU also checks for access violation. Errors detected by the IOMMU are access to a invalid page and access with protection violation. An invalid error happens when the DVMA virtual page does not have a valid physical page mapped to it. A protection error happens when the SBus master tries to write to a page that is marked as read-only. Both errors will be reported with the SBus error acknowledge to the device. The actual reporting of translation errors is device dependent.

17.2.2 Summary of Error Reporting

Table 17-1 summarizes the reporting of fatal errors detected in the reference platform system.

Table 17-1 Summary of Fatal Error Reporting in the Reference Platform

| Error Type | Type of Operation | System Action | CPU Action | Error Register |
|--------------------------|-----------------------------|---------------|------------|--|
| UPA address parity error | All | Reset | Reset | USC Port_Status Register USC Control Register |
| Master queue overflow | All | Reset | Reset | USC Port_Status Register |
| DTAG parity error | CPU IFetch, CPU LD/ST, DVMA | Reset | Reset | CC fault register |
| Coherence error | CPU IFetch, CPU LD/ST, DVMA | Reset | Reset | CC fault register |

Table 17-2 summarizes the reporting of non-fatal errors detected in the reference platform system.

Table 17-2 Summary of Non-Fatal Error Reporting in the Reference Platform

| Error Type | Type of Operations | Response to UPA ^[1] | S.F. Trap | Error Register ^[2] | SBus Action |
|---------------------------|--|--------------------------------|-----------|---------------------------------------|-------------------------------|
| E-S Data RAM Parity Error | CPU IFetch, CPU LD/ST, UPA snoop | Force bad ECC if UPA snoop | Yes | UltraSPARC-I AFSR/AFAR | No |
| E-S Tag RAM Parity Error | CPU IFetch, CPU LD/ST, UPA snoop | Force bad ECC if UPA snoop | Yes | UltraSPARC-I AFSR/AFAR | No |
| UPA UE | PIO read Memory read UPA interrupt | No | Yes | UltraSPARC-I AFSR/AFAR ^[3] | No |
| | PIO write | UE interrupt | No | U2S UE AFSR/AFAR ^[3] | No SBus cycle |
| | DVMA read | UE interrupt | No | U2S UE AFSR/AFAR ^[3] | SBus Error Ack ^[4] |
| | DVMA write | UE interrupt | No | U2S UE AFSR/AFAR ^[3] | SBus Data Ack |

Table 17-2 Summary of Non-Fatal Error Reporting in the Reference Platform

| Error Type | Type of Operations | Response to UPA ^[1] | S.F. Trap | Error Register ^[2] | SBus Action |
|-----------------------------|----------------------------|-----------------------------------|-----------|--|------------------------------------|
| UPA CE | PIO Read Memory read | No | Yes | UltraSPARC-I AFSR/AFAR | No |
| | PIO write | CE interrupt | No | U2S CE AFSR/AFAR | No |
| | DVMA read | CE interrupt | No | U2S CE AFSR/AFAR | SBus Data Ack |
| | DVMA write | CE interrupt | No | U2S CE AFSR/AFAR | SBus Data Ack |
| UPA Read Error, UPA Timeout | PIO Read Memory Read | UPA read error, UPA timeout | Yes | UltraSPARC-I AFSR/AFAR, USC Port Status Register | No |
| | DVMA Read | UPA read error, UPA timeout | No | No | SBus Error Ack ^[5] |
| | DVMA Write | No | No | No | SBus Data Ack |
| SBus Parity Error | PIO Read | Force bad ECC to UPA | Yes | UltraSPARC-I AFSR/AFAR, SBus control register | SBus Data Ack |
| | PIO Write | No | No | No | SBus Data/Error Ack ^[5] |
| | DVMA Virtual Address | No | No | No | SBus Error Ack ^[4] |
| | DVMA Read Data Phase (ETI) | No | No | No | SBus Data Ack ^[5] |
| | DVMA Write Data Phase | Force bad ECC to UPA | No | U2S SBus control register | SBus Data Ack |
| SBus Timeout | PIO read | UPA Timeout | Yes | UltraSPARC-I AFSR/AFAR | SBus Error Ack |
| | PIO write | SBus Asynchronous Error Interrupt | No | U2S SBus Asynchronous Error AFSR/AFAR | SBus Error Ack |
| | DVMA read DVMA write | No | No | No | SBus Error Ack ^[4] |

Table 17-2 Summary of Non-Fatal Error Reporting in the Reference Platform

| Error Type | Type of Operations | Response to UPA ^[1] | S.F. Trap | Error Register ^[2] | SBus Action |
|-------------------------|-------------------------|-----------------------------------|-----------|---------------------------------------|-------------------------------|
| SBus Error Ack | PIO read | UPA Read Error | Yes | UltraSPARC-I AFSR/AFAR | SBus Error Ack |
| | PIO write | SBus Asynchronous Error Interrupt | No | U2S SBus Asynchronous Error AFSR/AFAR | SBus Error Ack |
| | DVMA read DVMA write | No | No | No | SBus Error Ack ^[4] |
| SBus Late Error | PIO read | SBus Asynchronous Error Interrupt | No | U2S SBus Asynchronous Error AFSR/AFAR | SBus Data Ack |
| | PIO write | SBus Asynchronous Error Interrupt | No | U2S SBus Asynchronous Error AFSR/AFAR | SBus Data Ack |
| | DVMA read DVMA write | No | No | No | SBus Data Ack Late Error |
| IOMMU Translation Error | DVMA read DVMA write | No | No | No | SBus Error Ack ^[4] |

1. This assumes error interrupts are enabled.
2. Information logged in the AFSR/AFAR depends on the type of error and the state of the error status. Please refer to the *UltraSPARC-I Programmer's Reference Manual* and System Register Definition of this document for more details.
3. The UE may be caused by other errors in the system. Additional information is available in the UltraSPARC-I and U2S registers. See the ECC Error section for more details.
4. The handling of a SBus error acknowledge during the DVMA cycle is device dependent. The device should stop DVMA from the channel getting error ack and post an interrupt to the system.
5. The handling of a SBus parity error detected by the SBus device is device specific. The device should generate an interrupt to the system.

17.3 Unreported Errors

Certain error conditions are not reported by the system. Examples of these errors are listed below. Beware that the list is not intended to enumerate all possibilities.

- A write to a non-existing or disconnected UPA port. The IADD bit in the Port_Status Register will be set, but no error is reported
- A write to a read-only register in the USC or U2S is ignored
- A non-cached UPA write to memory. The IADD bit in the UPA Port_Status Register will be set but no error reported
- A read from a write-only register in the USC or U2S gets invalid data
- The UPA bus error or timeout during DVMA write is ignored
- Sending an interrupt to a non-existing UPA port or disconnected UPA port. The IPORT bit in the Port_Status Register will be set, but no error is reported

18.1 Introduction

This chapter briefly describes the JTAG Test Access Port (*JTAGTAP*) macro which implements the IEEE 1149.1-1990 Standard (also known as JTAG) Test Access Port on the ASICs in the UltraSPARC-I system. For more information see the complete specifications, “JTAG Test Access Port for UltraSPARC-I system ASICs.”

The *JTAGTAP* macro provides standardized serial access to the test features designed into the ASIC. It includes the JTAG state machine, the instruction register, a bypass register, a device identification register, and all of the decoding logic so as to simplify the designers task of implementing JTAG. Signals are provided for access to the mandatory boundary scan register, and to optional internal scan registers or other user defined test features (such as Built-In Self-Test).

The *JTAGTAP* macro requires five dedicated pads (these pins must *NOT* be used for any other purpose). These include the serial test data input (TDI), serial test data output (TDO), test mode select (TMS) and the test clock (TCK), and asynchronous reset (TRST_).

18.2 Features

Some features of the *JTAGTAP* macro:

- standard 16 bit instruction register
- 8 bits reserved for pre-defined instructions which are standard
- 8 bits available for up to 254 user defined test functions
- capable of sampling up to 14 status signals
- device identification register with 12 bits for chip ID and 4 bits for chip vintage
- bypass register
- provides all the signals required to access the boundary scan register
- provides all the signals required to access any internal scan registers
- capable of tri-stating all the chip outputs via a single instruction shift

18.3 TAP signals

The JTAG TAP uses the following inputs and outputs:

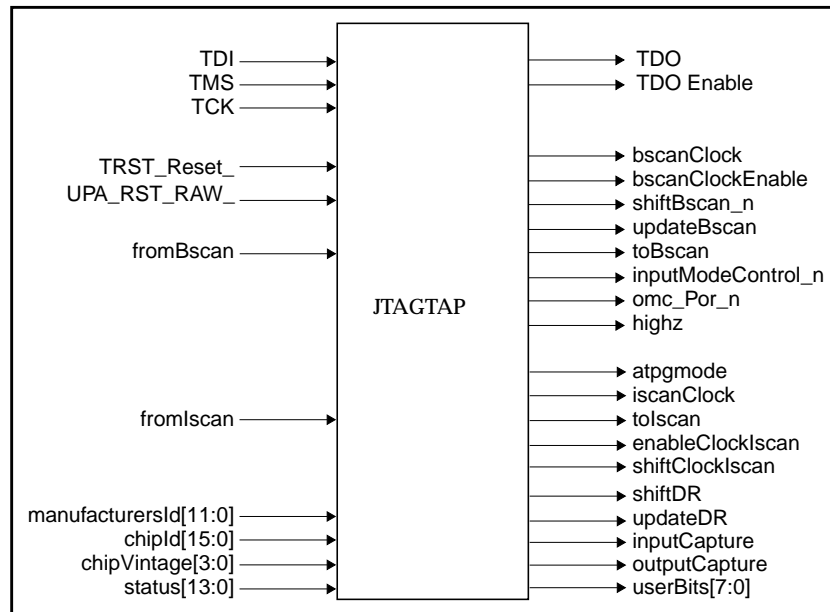


Figure 18-1 JTAG TAP Inputs and Outputs

Index



- A
- ACK 23
- Address/Data Path 189
- Address/Data path 179
- AFAR 31, 38
- AFSR 31
- AFSR/AFAR 38
- ARB_EN 38
- Arbiter (sbm_arb) 182
- B
- Boot Device 26
- Bus Block 16
- Bus Control 7, 19
- bus-control uni 19
- BY bit 196
- Bypass Mode 196
- C
- CE AFAR 34
- CE AFSR 33
- CE AFSR/AFAR 33
- CLK/CLK 2
- clock controller 4
- Coherency Protocol 172
- Consistent DVMA 201
- Consistent DVMA Read 202
- Consistent DVMA Write 202
- Control/Status Register 26
- Correctable Error Asynchronous Fault
- Status/Address Register 33
- Count and Limit registers 227
- Count Register 223
- counter 230
- CP b 196
- D
- Dispatcher FSM 220
- DMA 6, 36
- DMA Control 19
- DMA Data and Address Paths 18
- DMA Merge Buffer 21
- DMA Merge Buffer block 21
- DMA Writes 173
- DMA-control 19
- DVMA 1, 7, 32, 179, 190, 196, 201, 202
- DVMA Controller 186
- DVMA controller 179
- DVMA cycles 21, 193
- DVMA Error Conditions 188
- DVMA read 203, 204
- DVMA RW Buffer 179

- DVMA traffi 22
- DVMA transaction 187
- DVMA transactions 181
- DVMA types 187
- DVMA Write 202
- DVMA write 206
- DVMA writes 22
- E
- e UPA to the SBu 179
- ECC 213
- ECC Check 6
- ECC check logic 20
- ECC Control Register 30
- ECC Error Reporting 31
- ECC Generate/Check 20
- ECC Registers 30
- ECC unit 20
- ERRINT_EN 37
- Error Correcting Code 6
- Ethernet oscillator 230
- External Interrupt Concentrator 221
- External Interrupts 214
- F
- Fast SBus bit 190
- FAST_SBUS 37
- freezing 174
- Functional Overview 15
- I
- I/O cells 9
- I/O Driver Specifications 9
- IEEE 1149.1-1990 Standard 243
- IEEE P1496 1, 177
- IEEE P1496 specification 21
- IMPL 36
- independent timers 223
- Input/Output
- Memory Management Unit 1
- INR 212
- Internal Bus Block 16
- Internal Interrupts 213
- interrupt 215
- Interrupt Arbiter 219
- interrupt concentrato 23
- Interrupt Concentrator 214
- Interrupt Decoder 219
- Interrupt Dispatcher 220
- interrupt logic 22
- Interrupt Number Register 212
- Interrupt Number Registers 23
- Interrupt Ordering 190
- Interrupt Receiver 219
- Interrupt Types 213
- IOMMU 22, 179, 193, 194, 195, 200, 201
- IOMMU Demap 200
- IOMMU mapping 21
- J
- JTAG 24
- Jtag 243
- JTAG Control 7
- JTAG TA 7
- JTAG Test Access Port 243
- JTAGTAP 243
- L
- Level/Pulse 215
- Limit Register 223
- LRU algorithm 22, 203
- M
- MD 2
- MDU 7, 23, 207
- Merge Buffer 7, 173
- MMU_EN bit 194
- MODE bit 27
- Mondo Dispatch 209
- Mondo Dispatch Block 210
- Mondo Dispatch Unit 7, 23, 207, 209, 211

- Mondo Vector Format 211
- Mondo Vector Unit 230
- Mondo Vectors 211
- Mondo-Vector Dispatch unit 2
- N
- NACKed 23
- O
- OMMU 7
- P
- P_Reply requests 20
- Packet Builder FSM 220
- packet-switched main system bus 6
- Partial Write Data 176
- Pass-Through Mode 196
- Performance Counters 24
- Physical Address of ECC Registers 30
- Pin assignments 9
- PIO and DMA cycles 15
- PIO Control 7
- PIO Controller 183
- PIO controller 179
- PIO cycle 19
- PIO Decoder 19
- PIO Error Conditions 185
- PIO requests 19, 21
- PIO RW Buffer 179
- PIO RW Buffer 179
- PIO transaction 179, 184
- PIO transactions 177, 183
- PIO-control 19
- PIO-control unit 20
- Priority 216
- protection error 200
- Pulse Interrupts 215
- R
- REGFILE cells 2
- Register accesses 25
- REV 36
- RIC 4, 23, 209
- RW Buffer 179
- S
- S_Reply 20
- SBM 7, 21, 22, 177, 179, 180, 183
- SBM Control Register 190
- SBM reset 182
- SBM resets 179
- sbm_dvma 186
- sbm_pio 183
- sbm_toc 181
- SBu 21
- SBus 4, 6, 22, 38, 179, 207
- SBus AFSR 39
- SBus Asynchronous Fault Status 38
- SBus clock 21
- SBus Control Register 34, 35
- SBus DVMA cycles 22
- SBus Interface Block 7
- SBus Interface Signals 12
- SBus IOMMU 193
- SBus master 195
- SBus Master/Slave Slots 177
- SBus Module 34, 177, 213
- SBus Module Functional Description 181
- SBus Registers 34
- SBus Slave Only Slo 177
- SBus Slot Configuration Register 34, 40
- SBus virtual address 203
- SCache 7, 21, 22
- SCache logic 22
- Scan/JTAG 12
- SLAVIO 26, 40
- SPARC V 2, 7
- SSCR 34, 196
- SSCR controls 196
- Stream DVM 201

- Stream Read 203
- Stream Write 204
- Streaming Cache 22, 179
- streaming cache 204, 206
- Streaming Cache Flush 205
- Streaming Cache Invalidation 204
- Streaming Cache Management 204
- SYSIO 1
- system coherency 173
- system controller 29
- T
- TAP signals 244
- timer 230
- Timer Counters 7, 213
- Timer unit 224
- Timer/Counters 24
- TLB Data RAM 200
- TLB flush 200
- TLB Initialization 200
- Total Pin Count 14
- Translation Errors 200
- Translation Look-Aside Buffer 193
- Translation Storage Buffer 22, 193, 196
- Translation Table Entry 25, 197
- treame DVMA 203
- TSB Base Address Register 198
- TSB entry 200
- TSB Lookup 198
- TTE 25
- TTE Entry Address 198
- U
- U2S 1, 2, 3, 5, 27, 29, 174, 193, 202, 203, 207, 209, 211
- U2S Block Diagram 16
- U2S clock 27
- U2S Control Register Definition 26
- U2S control sections 19
- U2S DMA 19
- U2S Executive 2
- U2S Function Blocks 6
- U2S pins 4, 23
- UE AFAR 32
- UE AFSR 32
- UE AFSR/AFAR 31
- UE and CE errors 20
- UltraSPARC 1, 171, 172, 243
- UMS 6, 19
- Uncorrectable Error
- Asynchronous Fault Status 31
- UPA 6, 27, 179, 211
- UPA clock 27
- UPA coherence algorithm 23
- UPA Configuration Register 29
- UPA Data Bus 211
- UPA interconnect 196
- UPA interface 177
- UPA Interface Blocks 6
- UPA Interface Signals 10
- UPA Interrupt 209
- UPA Master/Slave 19
- UPA packe 19
- UPA Read-To-Own 21
- UPA Registers 28
- UPA Reply 20
- UPA request 29
- UPA slave 19
- UPA/memory bandwidth 203
- UPA-64 177
- URP 6
- USC blocks 21
- V
- valid information 197
- virtual addres 206
- virtual page 197
- W
- WAKEUP_EN 37