

*USC User's Manual*

*Sun Microelectronics*

## *USC 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



<b>1. Overview.....</b>	<b>1</b>
1.1 Introduction.....	1
1.2 USC Summary .....	1
1.3 Package and Die .....	3
1.4 Block Diagrams.....	4
<b>2. Pin List and Description .....</b>	<b>7</b>
2.1 UPA Interface Signals .....	8
2.2 Memory Interface Signals.....	9
2.3 XB1 Interface Signals.....	9
2.4 EBus Interface .....	10
2.5 Miscellaneous Signals .....	11
2.6 Power and Ground.....	12
2.7 Total Pin Count.....	12
<b>3. Functional Description.....</b>	<b>13</b>
3.1 Introduction.....	13
3.2 UPA Master ID Assignment .....	13
3.3 USC Queues .....	14
3.4 UPA Port Interface (PIF).....	14
3.5 Data Path Scheduler.....	15
3.6 Memory Controller .....	15
3.7 EBus Interface .....	16
3.8 Code Structure and Hierarchy .....	17
<b>4. Programming Model.....</b>	<b>21</b>
4.1 System Controller Registers.....	21

<b>5. Packet Handling</b> .....	45
5.1 Introduction.....	45
5.2 USC Transaction Table .....	45
5.3 Coherent Transactions .....	48
5.4 Noncached Transactions.....	52
5.5 Interrupts .....	54
5.6 Flow Control.....	54
5.7 Blocking Conditions.....	55
5.8 Class Symmetry .....	58
5.9 Presence Detection.....	58
5.10 DATA_STALL.....	59
5.11 Internal Arbitration Priorities .....	60
5.12 Concurrency .....	61
5.13 Reserved P_Replies .....	62
5.14 Error Handling.....	62
5.15 Timing Diagrams.....	63
<b>6. Memory System</b> .....	73
6.1 Introduction.....	73
6.2 SIMMs .....	73
6.3 Block Diagram.....	74
6.4 Addressing .....	76
6.5 Refresh.....	79
6.6 Timing Diagrams.....	80
<b>7. Resets</b> .....	107
7.1 Introduction.....	107
7.2 Reset Signals.....	108
7.3 Reset Operation.....	110
7.4 Sleep and Wakeup.....	111
<b>8. EBus Interface</b> .....	113
8.1 Introduction.....	113
8.2 Functional Description.....	113
8.3 Timing Diagrams.....	115
<b>9. Miscellaneous Items</b> .....	119
9.1 Introduction.....	119
9.2 Process Monitor .....	119
9.3 Spare Gate Macros.....	120
9.4 Debug Pins.....	120
9.5 Performance Counters .....	120

## List of Figures



USC Block Diagram .....	4
System Controller in the UltraSPARC-I Reference Platform .....	5
Hierarchy Diagram Convention .....	18
Top-Level Hierarchy .....	19
sc_top Hierarchy .....	19
PC1 Field Timing Effects .....	42
Best-Case PRequest-to-PRequest Timing (ABus0 to ABus0) .....	63
Best-Case PRequest-to-SRequest Timing (ABus0 to ABus0) .....	64
Best-Case PRequest-to-SRequest Timing (ABus0 to ABus1) .....	64
Best-Case Timing for Noncached Single Read, UPA64 -> UPA128 .....	65
Best-Case Timing for Noncached Block Read, UPA64 -> UPA128 .....	65
Best-Case Timing for Noncached Single Write, UPA128 -> UPA64 .....	66
Best-Case Timing for Noncached Block Write, UPA128 -> UPA64 .....	66
Best-Case Timing for Noncached Single Read, UPA64 -> UPA64 .....	67
Best-Case Timing for Noncached Block Read, UPA64 -> UPA64 .....	67
Best-Case Timing for Noncached Single Write, UPA64 -> UPA64 .....	68
Best-Case Timing for Noncached Block Write, UPA64 -> UPA64 .....	68
Best-Case Timing for Noncached Single Read, UPA128 -> UPA64 .....	69
Best-Case Timing for Noncached Block Read, UPA128 -> UPA64 .....	69
Best-Case PRequest-to-Memory Request Timing, Read (Fast Path) .....	70
Best-Case PRequest-to-Memory Request Timing (Normal Path) .....	71
Memory System Interconnection .....	74
Addressing .....	76
Basic Read/Write Timing .....	80
Basic Refresh Timing .....	81
Best-Case UPA-to-Memory Timing (Fast Path) .....	82
Best-Case UPA-to-Memory Timing (Normal Path) .....	82

Default Memory CPU Read Timing .....	85
Default Memory CPU Write Timing .....	86
Default Memory U2S Read Timing .....	86
Default Memory U2S Write Timing .....	87
Default Refresh Timing .....	87
83.3-MHz CPU Read Timing .....	88
83.3-MHz CPU Write Timing .....	88
83.3-MHz U2S Read Timing .....	89
83.3-MHz U2S Write Timing .....	89
83.3-MHz Refresh Timing .....	90
71.4-MHz CPU Read Timing .....	90
71.4-MHz CPU Write Timing .....	91
71.4-MHz U2S Read Timing .....	91
71.4-MHz U2S Write Timing .....	92
71.4-MHz Refresh Timing .....	92
66.7-MHz CPU Read Timing .....	93
66.7-MHz CPU Write Timing .....	93
66.7-MHz U2S Read Timing .....	94
66.7-MHz U2S Write Timing .....	94
66.7-MHz Refresh Timing .....	95
62.5-MHz CPU Read Timing .....	95
62.5-MHz CPU Write Timing .....	96
62.5-MHz U2S Read Timing .....	96
62.5-MHz U2S Write Timing .....	97
62.5-MHz Refresh Timing .....	97
55.5-MHz CPU Read Timing .....	98
55.5-MHz CPU Write Timing .....	98
55.5-MHz U2S Read Timing .....	99
55.5-MHz U2S Write Timing .....	99
55.5-MHz Refresh Timing .....	100
50-MHz CPU Read Timing .....	100
50-MHz CPU Write Timing .....	101
50-MHz U2S Read Timing .....	101
50-MHz U2S Write Timing .....	102
50-MHz Refresh Timing .....	102
CPU Read Timing .....	103
CPU Write Timing .....	103
U2S Read Timing .....	104
U2S Write Timing .....	104
Refresh Timing .....	105
Resets .....	107

## *List of Figures*

Internal Read Timing .....	115
Internal Write Timing .....	116
External EBus Read Timing .....	116
External EBus Write Timing .....	117
Process Monitor .....	119

*USC User's Manual*



## List of Tables



UPA Interface Signals .....	8
Memory Interface Signals .....	9
XB1 Interface Signals .....	9
EBus Interface Signals .....	10
Miscellaneous Signals .....	11
Power and Ground .....	12
Pin Count Breakdown by Function .....	12
Pin Count Breakdown by Input/Output/IO .....	12
Master ID .....	13
USC Queue Lengths and Depths .....	14
Source and Destination Designators .....	17
RTL Directories .....	18
USC Internal Address Space .....	22
Address Assignments of USC Address and Data Registers.....	23
SC_Address Register .....	23
SC_Data Register .....	24
Register Access Type Codes .....	25
SC_Control Register .....	26
USC Reset Strategy .....	27
SC_ID Register .....	30
SC_Perf0 Register .....	30
SC_Perf1 Register .....	31
SC_PerfShadow Register .....	31
Performance Monitor Control Register .....	32
USC Performance Counter 0 Event Sources .....	32
USC Performance Counter 1 Event Sources .....	33
P0_Config Register .....	34

*USC User's Manual*

P0_Status Register .....	35
SYSIO_Config Register .....	36
SYSIO_Status Register .....	37
SC Port_Status Register MC0Q and MC1Q Init Table.....	37
MC_Control0 Register .....	38
SIMMPresent Encoding .....	39
RefInterval Table for SIMMs Using 4 MB (1024 Rows/16 ms), 16 MB (4096 Rows/64 ms), or 64 MB (8192 Rows/128 ms) .....	39
MC_Control Register 1 .....	40
MC_Control1 Values as a Function of Operating Frequency .....	43
USC Transaction Table .....	46
USC Master Queue Sizes .....	54
Resource Usage for Cached Transactions .....	57
DATA_STALL Assertion .....	59
SIMMs Supported by the USC .....	73
PA[29:28] to $\overline{\text{RAS}}$ Mapping.....	76
Memory Address Map .....	78
MC_Control1 Timing Values .....	84
Reset Table .....	111
EBus Register Map .....	114

## 1.1 Introduction

This document describes the Uniprocessor System Controller (USC<sup>[1]</sup>) used on the UltraSPARC™ - I Reference Platform. The USC is a specially designed ASIC system controller. The UltraSPARC-I Reference Platform System is an economical, high-performance uniprocessor system.

## 1.2 USC Summary

The USC's primary function is to regulate the flow of requests and data throughout the entire system. The USC's secondary function is to control the resets going to all UltraSPARC Port Architecture (UPA) clients.

### 1.2.1 Features

#### 1.2.1.1 UPA Ports

- The USC implements three UPA ports on two UPA address buses, as shown in Figure 1-2
- Address port 0 is a full implementation that supports two master/slaves, the processor and the U2S, UPA-to-SBus Interface
- Address port 1 is a UPA64S subset implementation that supports slave-only graphics. The P\_Reply and S\_Reply for the graphics slave are subsets of the full P\_Reply and S\_Reply, as defined in the UPA specification. The address bus is only 29 bits wide instead of 35 bits and does not have a parity bit

---

1. In previous documentation, the USC was named SC\_UP.

### ***1.2.1.2 Data Path Control***

The USC controls the XB1 (Crossbar Switch) and controls the flow of data through the entire system.

### ***1.2.1.3 Memory Interface***

The USC contains the memory controller. It can control up to eight SPARC 20-type DRAM SIMMs. The memory controller will only work with the new-style DSIMMs which use 60-ns DRAMs; it will not work with the old-style DSIMMs which use 80-ns DRAMs.

The memory controller supports 16-MB, 32-MB, 64-MB, and 128-MB SIMM modules built out of 4-MB, 16-MB, or 64 MB DRAM devices.

### ***1.2.1.4 Resets***

The USC controls and generates a number of resets for the system. It accepts reset inputs from the RIC (Reset/Interrupt/Clock Controller) and forwards them to other parts of the system.

### ***1.2.1.5 EBus Interface***

Since the USC has no data bus, an eight-bit asynchronous EBus interface is provided to allow reading and writing of the USC's internal registers. EBus is controlled by the SLAVIO (Slave I/O controller) chip.

### ***1.2.1.6 JTAG Interface***

The USC provides a JTAG interface for full-chip scan, which is used only for ATPG and in system interconnect testing. The USC's boundary is shadowed to allow for board-level test.

## ***1.3 Package and Die***

Currently the USC is packaged in a 225-pin BGA (ball grid array) package.

### ***1.3.1 Frequency of Operation***

The USC will operate at a maximum frequency of 100 MHz (10-ns cycle time). It is a completely synchronous, edge-triggered register-based design which uses only the rising edge of the clock to update the flip flops.

The USC also contains a phase-locked loop (PLL) to remove the skew introduced by the internal clock distribution network.

## 1.4 Block Diagrams

A high-level block diagram of the USC is shown in Figure 1-1, with abbreviated signal names.

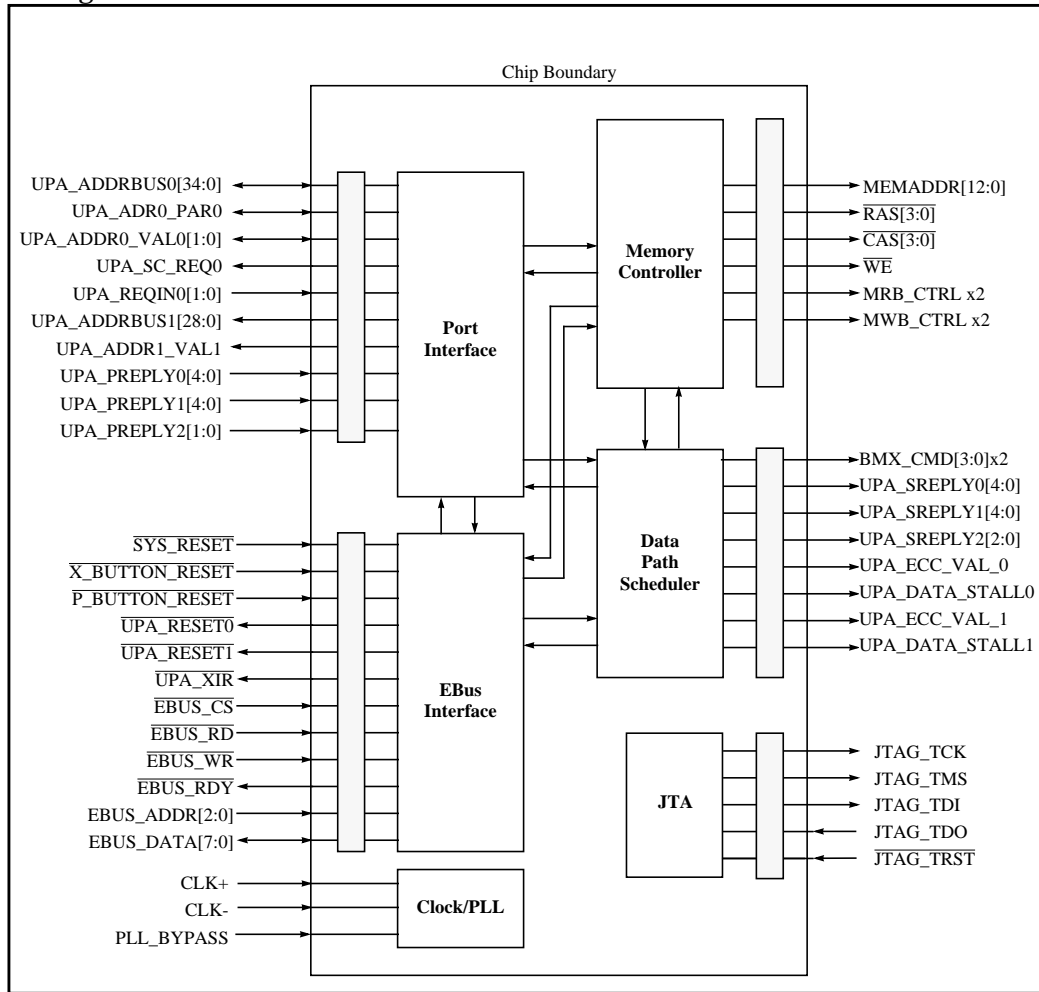


Figure 1-1 USC Block Diagram

## 1. Overview

Figure 1-2 is a typical application diagram employing the USC. It also shows how the USC is used in the reference platform.

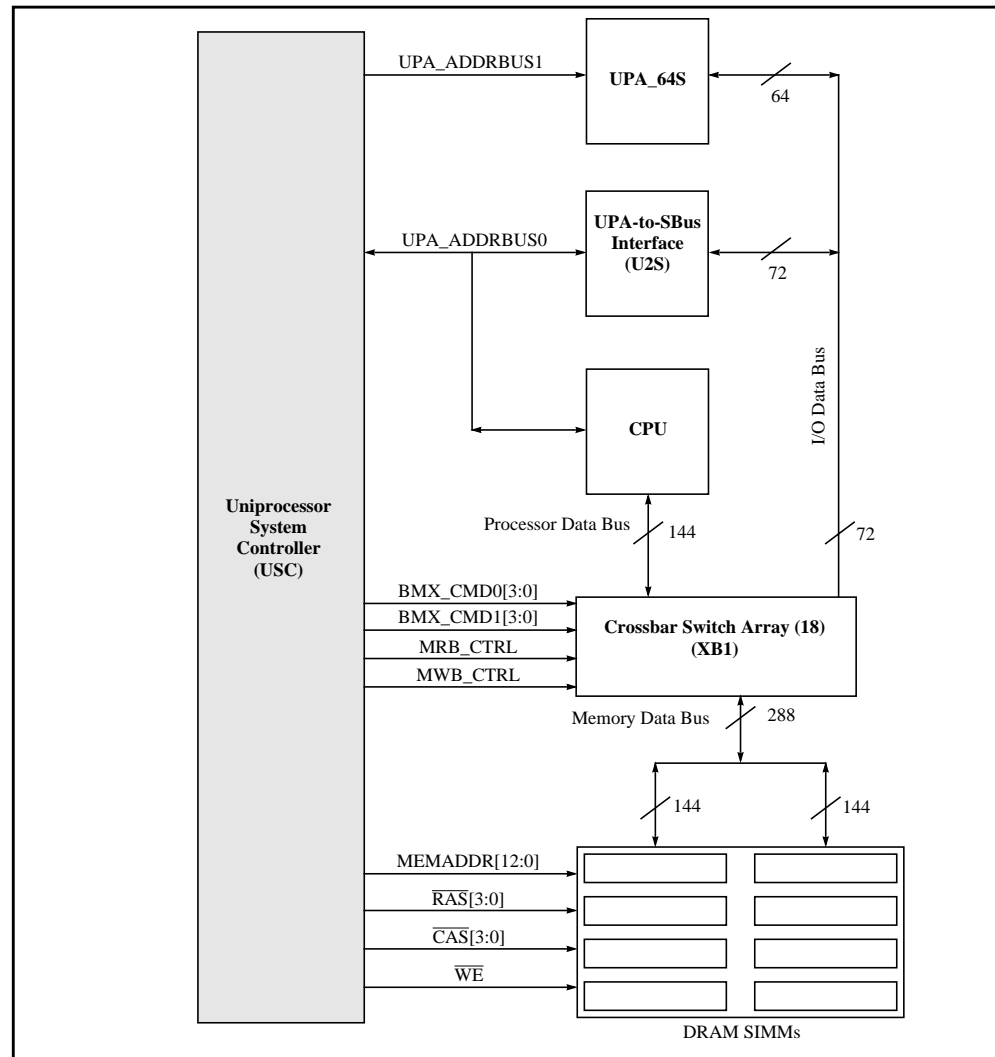



Figure 1-2 System Controller in the UltraSPARC-I Reference Platform





## *Pin List and Description*

---

2 

This chapter lists all of the USC's signal pins. See the ATT HL400C data book for a description of the buffer types.

## 2.1 UPA Interface Signals

Table 2-1 UPA Interface Signals

Signal	Pin Count	I/O	Buffer Type	Description
$\overline{\text{UPA\_ADDRBUS0}}[34:0]$	35	I/O	BB02Z24T	Address bus 0 (processor/U2S)
UPA_ADR0_PAR	1	I/O	BB02Z24T	Parity for address bus 0
UPA_ADDR0_VAL[1:0]	2	I/O	BB02Z24T	[0] = processor, [1] = U2S
UPA_SC_REQ0	1	O	BOZ24T	SC request for address bus 0
UPA_REQIN0[1:0]	2	I	BIN02T	Client address bus 0 arbitration requests: [0] = processor, [1] = U2S
UPA_ADDRBUS1[28:0]	29	O	BOZ24T	Address bus 1 - UPA64S (slave only)
UPA_ADDR1_VAL1	1	O	BOZ24T	Address valid for the UPA64S
UPA_SREPLY0_[4:0]	5	O	BOZ24T	S_Reply for the processor
UPA_SREPLY1_[4:0]	5	O	BOZ24T	S_Reply for the U2S
UPA_SREPLY2_[2:0]	3	O	BOZ24T	S_Reply for the UPA64S
UPA_PREPLY0_[4:0]	5	I	PINA02T	P_Reply from the processor
UPA_PREPLY1_[4:0]	5	I	PINA02T	P_Reply from the U2S
UPA_PREPLY2_[1:0]	2	I	PINA02T	P_Reply from the UPA64S
$\overline{\text{UPA\_RESET0}}$	1	O	BOZ24T	Reset for processor, tied to the U2S's $\overline{\text{UPA\_ARB\_RESET}}$
$\overline{\text{UPA\_RESET1}}$	1	O	BOZ24T	Reset for the U2S
$\overline{\text{UPA\_XIR}}$	1	O	BOZ24T	XIR reset for the processor only
UPA_ECC_VAL_0	1	O	BOZ24T	ECC valid for the processor
UPA_ECC_VAL_1	1	O	BOZ24T	ECC valid for the U2S
UPA_DATA_STALL0	1	O	BOZ24T	Stall data to the processor
UPA_DATA_STALL1	1	O	BOZ24T	Stall data to the U2S
Total UPA port interface signals	103			

## 2.2 Memory Interface Signals

Table 2-2 Memory Interface Signals

Signal	Pin Count	I/O	Buffer Type	Description
MEMADDR[12:0]	13	O	P4OZ24T	Row/column address
$\overline{\text{RAS}}[3:0]$	4	O	BOZ24T	$\overline{\text{RAS}}$ per SIMM pair
$\overline{\text{CAS}}[3:0]$	4	O	BOZ24T	$\overline{\text{CAS}}$ per SIMM
$\overline{\text{WE}}$	1	O	BOZ24T	Write enable
Total memory interface	22			

## 2.3 XB1 Interface Signals

Table 2-3 XB1 Interface Signals

Signal	Pin Count	I/O	Buffer Type	Description
BMX_CMD0[3:0]	4	O	BOZ24T	Command to XB1 crossbar array of 18 devices*
MRB_CTRL0	1	O	BOZ24T	Fill the XB1 read buffer
MWB_CTRL0	1	O	BOZ24T	Drain the XB1 write buffer
BMX_CMD1[3:0]	4	O	BOZ24T	Duplicate of BMX_CMD0
MRB_CTRL1	1	O	BOZ24T	Duplicate of MRB_CTRL0
MWB_CTRL1	1	O	BOZ24T	Duplicate of MWB_CTRL0
Total XB1 interface	12			

## 2.4 EBus Interface

Table 2-4 EBus Interface Signals

Signal	Pin Count	I/O	Buffer Type	Description
EBUS_DATA[7:0]	8	I/O	B5N3Z10T	Data in and data out pins - TTL levels; 5-V tolerant
$\overline{\text{EBUS\_CS}}$	1	I	B5IN03T	Chip select for the USC on the EBus; 5-V tolerant
EBUS_ADDR[2:0]	3	I	B5IN03T	EBus address; 5-V tolerant
$\overline{\text{EBUS\_RDY}}$	1	O	B5OZ10T	EBus ready to the SLAVIO; 5-V tolerant
$\overline{\text{EBUS\_WR}}$	1	I	B5IN03T	Indicates write on the EBus; 5-V tolerant
$\overline{\text{EBUS\_RD}}$	1	I	B5IN03T	Indicates read on the EBus; 5-V tolerant
Total EBus interface signals	15			

## 2.5 Miscellaneous Signals

Table 2-5 Miscellaneous Signals

Signal	Pin Count	I/O	Buffer Type	Description
CLK+	1	I	BIED03T	System clock (PECL differential)
CLK-	1	I	BIED03T	System clock (PECL differential)
$\overline{\text{SYS\_RESET}}$	1	I	PINX06T	Power-on reset; pulldown
$\overline{\text{P\_BUTTON\_RESET}}$	1	I	BIN06T	POR button reset
$\overline{\text{X\_BUTTON\_RESET}}$	1	I	BIN06T	XIR button reset
PLL_BYPASS	1	I	BIN06T	Bypass the internal PLL
JTAG_TDI	1	I	PINA02T	Test data input
JTAG_TDO	1	O	B5OZ10T	Test data output; 5-V output
JTAG_TCK	1	I	BIN02T	Scan clock
JTAG_TMS	1	I	PINA02T	Test mode select; pullup
$\overline{\text{JTAG\_TRST}}$	1	I	PINA02T	Reset the TAP controller; pullup
DEBUG[3:0]	4	O	BON10T	Debug pins
PM_OUT	1	O	BON10T	Process monitor output
Total miscellaneous signals	16			

## 2.6 Power and Ground

Table 2-6 Power and Ground

Signal	Pin Count	Description
V <sub>DD</sub>	22	+3.3 V
V <sub>SS</sub>	31	Ground
PLL_VDD	1	Power for PLL
PLL_GND	1	Ground for PLL
V <sub>CC</sub>	1	5-V reference for 5-V tolerant inputs
Total power and ground	56	

## 2.7 Total Pin Count

Table 2-7 Pin Count Breakdown by Function

Signal Group	Total Signals
UPA port interfaces	103
Memory interface	22
XB1 interface	12
EBus interface	15
Miscellaneous	16
Power and ground	56
Spares	1
USC total	225

Table 2-8 Pin Count Breakdown by Input/Output/IO

Type	Total Signals
Inputs	30
Outputs	92
Bidirectionals	46
Total USC signals	168
Power and ground	56
Spares	1
Total	225

### 3.1 Introduction

This chapter is a brief overview of the internal structure of the USC. It explains how functionality is divided inside the USC, and provides an overview of the structure of the Verilog source code.

The USC is divided into four major blocks:

1. Port Interface (PIF)
2. Data Path Scheduler (DPS)
3. Memory Controller (MC)
4. EBus Interface (EB)

These functions are described in further detail in the following sections.

### 3.2 UPA Master ID Assignment

The UPA master IDs are hardwired into the USC. The master ID numbers are shown in Table 3-1.

Table 3-1 Master ID

Master ID[4:0]	Client
5'b00000	CPU0
5'b00001	CPU1 (MP only)
5'b11111	U2S

Since the USC is the uniprocessor system controller, the Reference Platform can have only one CPU. In this design, two masters/slaves are CPU0 and U2S. CPU1 is kept for dual-processor system design. Since the UPA64S client fast frame buffer (FFB) is a slave-only device, it does not have a master ID.

### 3.3 USC Queues

The lengths of some important queues are listed in Table 3-2.

Table 3-2 USC Queue Lengths and Depths

Queue Name	Queue Depth in 2-Cycle Packets	Notes
MRQ0/CPU	1	Master request queue for CPU class 0
MRQ1/CPU	4	Master request queue for CPU class 1
MRQ/U2S	2	Master request queue for the U2S

### 3.4 UPA Port Interface (PIF)

The PIF implements the three UPA ports and two UPA address buses. It is responsible for receiving packets, decoding their destinations, and forwarding packets to the proper destinations. Cached transactions are typically forwarded to the memory controller. Noncached transactions are typically forwarded to the proper slave.

The UPA address bus 0 has two clients: the processor and the U2S.

The UPA address bus 1 implements the UPA64S subset only. It supports only a single graphics slave. This bus is output only, not bidirectional, and the USC is always the master of this bus. In addition, the graphics slave will only generate and receive truncated P\_Reply and S\_Reply.

The PIF controls the arbitration on the UPA address bus 0. There are two other masters on this bus, the processor and the U2S. The PIF uses a distributed round-robin algorithm as described in the “UPA Interconnect Architecture” document.

The PIF also receives all P\_Replies from UPA clients.

The PIF has the responsibility of maintaining coherence in the system between the processor's cache, main memory, and the U2S's merge buffer.



The PIF contains three sets of the following registers (one for each UPA port):

- SC\_Port\_Config registers
- SC\_Port\_Status registers

These registers are described in further detail in the UPA Reference Platform System document.

The operation of the PIF is described in further detail in Chapter 5, “Packet Handling.”

## 3.5 Data Path Scheduler

The Data Path Scheduler (DPS) is responsible for regulating the flow of data throughout the system. It generates the following:

- XB1 Crossbar Switch commands
- S\_Replies for all clients
- UPA\_DATA\_STALL signals
- UPA\_ECC\_VAL signals

The DPS contains no software-visible registers. The operation of the DPS is described in further detail in Chapter 5, “Packet Handling.”

## 3.6 Memory Controller

The memory controller (MC) is responsible for controlling the SIMMs. It performs the following functions:

- Generates timing for read, write, and refresh
- Converts the physical address in the UPA packet into row and column addresses
- Maintains the refresh timer
- Controls loading and unloading of data from the XB1 read and write buffers

The PIF forwards memory requests to the MC. The MC communicates with the DPS to schedule delivery of data.

The MC contains the following registers:

- MC\_Control0
- MC\_Control1

These registers are described in further detail in the UPA reference platform system document.

The operation of the MC and the memory subsystem is described in further detail in Chapter 6, "Memory System."

### **3.7 EBus Interface**

The EBus Interface (EB) implements an interface to the EBus, an asynchronous eight-bit interface controlled by the SLAVIO (Slave I/O Controller, STP2001). Since the USC contains no data path, all reading and writing of internal registers has to take place via the EBus. Since all internal registers are 32 bits wide, the EB has to perform packing and unpacking.

The EB is the only block which can be used in both the USC and MP without change. The EB block implements reset logic and contains a number of global registers.

- SC\_Control register for controlling resets and logging reset status
- SC\_ID register, which contains the USC's JEDEC ID number implementation and version numbers, and the number of UPA ports that the chip supports
- Performance counters: SC\_Perf\_Ctrl, SC\_Perf1, and SC\_Perf2. These counters can be configured to count various events for performance analysis
- SC\_Debug register: The USC has four debug pins, DEBUG[3:0], which allow some internal state to be visible on these pins for debug purposes. The SC\_Debug register allows the user to select which state will be visible on the external pins

The global registers are described in further detail in the UPA reference platform system document. The operation of the EB is described in further detail in Chapter 7, "Resets" and Chapter 8, "EBus Interface."

## 3.8 Code Structure and Hierarchy

### 3.8.1 Signal Naming Conventions

In an effort to improve the readability of the code, all the internal signal names in the USC follow a standard convention:

<source><destination>\_<signal name>

Source and destination are a single upper-case character.

Table 3-3 Source and Destination Designators

Source/Destination Block	Letter Designator
External signal	E
Port interface	P
Data path scheduler	D
Memory controller	M
EBus interface	B
Coherence controller (MP only)	C
Multidrop destination	X

Some examples:

1. PM\_ReqVal is an internal signal going from the Port Interface to the Memory Controller.
2. EP\_ADDR0[34:0] is a bus coming in from the external system and going to the Port Interface.
3. BX\_HardReset is a signal coming from the EBus block and going to multiple blocks.

### 3.8.2 Structure and Hierarchy

The Verilog for each major subblock of the device is contained in a separate directory. All source code is maintained under CDMS.

Table 3-4 RTL Directories

Block	Directory
Top level	\$CDMS_LOCALDIR/asic/sc/top_level/rtl
Port interface	\$CDMS_LOCALDIR/asic/sc/pif/rtl
Data path scheduler	\$CDMS_LOCALDIR/asic/sc/dps/rtl
Memory controller	\$CDMS_LOCALDIR/asic/sc/mc/rtl
EBus interface	\$CDMS_LOCALDIR/asic/sc/eb/rtl

All boxes in the hierarchy diagrams in the remaining sections have the following convention.

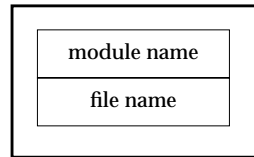


Figure 3-1 Hierarchy Diagram Convention

### 3.8.2.1 Top Level

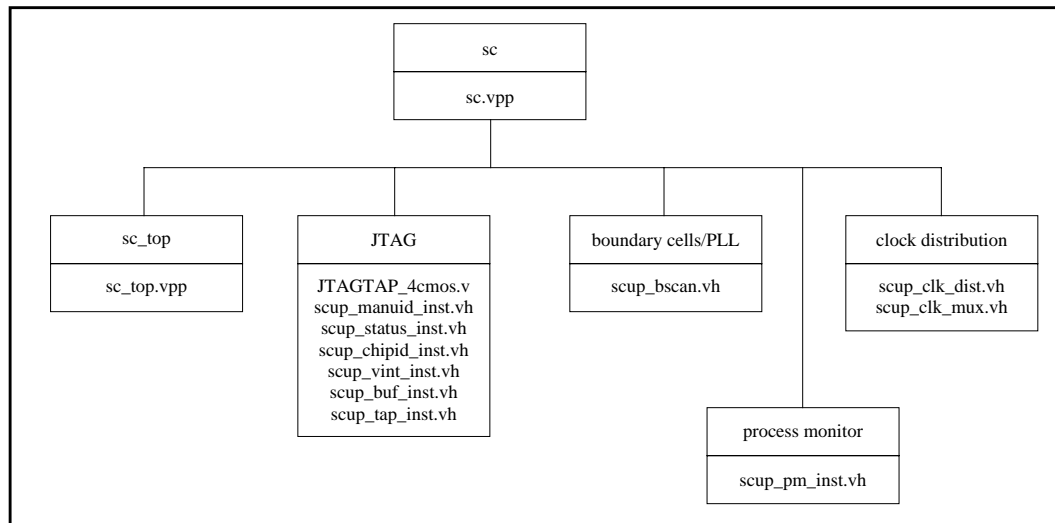


Figure 3-2 Top-Level Hierarchy

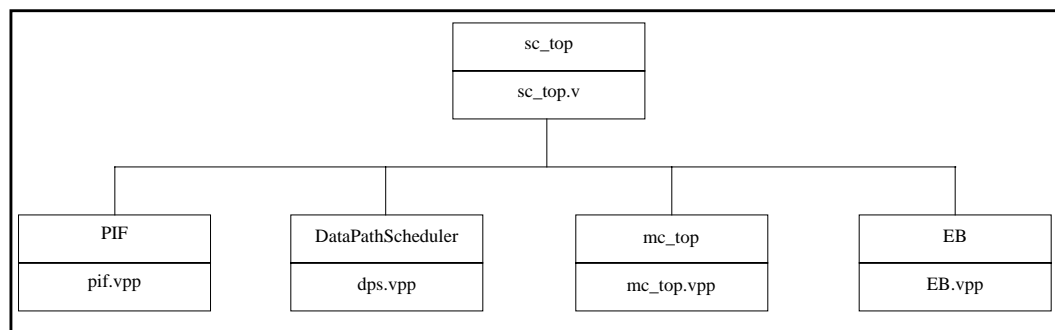


Figure 3-3 sc\_top Hierarchy



This chapter describes the address partitioning and software-visible registers and their respective functionality. The physical address associated with each of these registers is listed along with a description of the register.

---

**Note:** Registers which are designated as write-only may be read, but the data returned is *UNDEFINED*. No error is reported for such an access. Software should never rely on the value returned. Writes to read-only registers have no effect. No error is reported for such an access.

---

## 4.1 System Controller Registers

The system controller (USC) internal address map is shown below. Note that the addresses specified are only eight bits wide. This is because the addresses are not system addresses, but USC internal addresses. The USC is not addressed directly by software, rather the internal address map is addressed indirectly.

For all the registers that follow, not all 32 bits are specified. If a field is not specified explicitly, then it is considered *RESERVED*. Writes to *RESERVED* fields are ignored, and data read from a *RESERVED* field is always 0. Reads from an unimplemented USC internal addresses return *UNDEFINED* data. Writes to unimplemented USC internal addresses have unknown effects, as the address may wrap to an implemented address.

Table 4-1 USC InternalAddress Space.

Table 4-1 USC Internal Address Space

Internal Address	Register Name	Associated Port
0x00	SC_Control	USC overall
0x04	SC_ID	USC overall
0x08	SC_Perf0	USC overall
0x0C	SC_Perf1	USC overall
0x10	SC_PerfShadow	USC overall
0x20	SC_Perf_Ctrl	USC overall
0x30	SC_Debug_Pin_Ctrl	USC overall
0x40	P0_Config	Processor 0
0x44	P0_Status	Processor 0
0x48	P1_Config	Processor 1 <sup>[1]</sup>
0x4C	P1_Status	Processor 1 <sup>[1]</sup>
0x50	SYSIO_Config	U2S
0x54	SYSIO_Status	U2S
0x58	Reserved	Reserved
0x5C	Reserved	Reserved
0x60	MC_Control0 (USC)	Memory Controller
0x64	MC_Control1 (USC)	Memory Controller

1. For future multiple processor extension.

### 4.1.1 USC System Addresses

The USC requires two system addresses be allocated to it. The first system address is for the SC\_Address Register. This needs to be only a single byte. The second system address is for the SC\_Data Register. This must be a 32-bit word address, but it is accessed by software as four individual bytes.

The access to the SC\_Address Register and the SC\_Data Register is done through the SLAVIO generic port. Some address bits in the generic port address space are decoded to select the USC registers.



## 4. Programming Model

Table 4-2 shows the addresses of these two registers.

Table 4-2 Address Assignments of USC Address and Data Registers

Physical Address	Register
0x1FF.F130.0000	SC_Address register
0x1FF.F130.0004	SC_Data register

### 4.1.1.1 SC\_Address Register

The SC\_Address Register is a byte-access register. Table 4-3 gives the values software would write into this register to access other internal registers. The SC\_Address Register can be read and written.

### 4.1.1.2 SC\_Data Register

Table 4-3 SC\_Address Register

Field	Bits	PupState <sup>[1]</sup>	Description	Type
Address	7:0	X	Address pointer to the USC internal register map	R/W

1. PupState = Power-up state

Software accesses the SC\_Data Register with four consecutive atomic-byte accesses. Note that since atomicity of the four accesses is not guaranteed by the hardware, software must guarantee this by the use of a mutex lock or other such mechanism.

Since the access to this register is really an indirect access to another register in the USC, software must take care to access the bytes in the appropriate order.

---

**Note:** Byte ordering within the SC\_Data Register is big-endian (for example: byte 0 corresponds to bits 31:24).

---

For read and write accesses, the order is byte 0, followed by byte 1, then byte 2, then byte 3. Reading byte 0 causes the entire word to transfer to a holding register for subsequent reading, while writing byte 3 actually causes the 32-bit write to take place.

Table 4-4 SC\_Data Register

Field	Bits	PupState	Description	Type
Data	31:0	X	Writing byte 3 initiates the indirect write, reading byte 0 initiates the indirect read.	R/W

### 4.1.1.3 USC Indirect Addressing

To access any of the USC internal registers, software must first write the value of the internal address into the SC\_Address Register. Subsequent accesses to the SC\_Data Register actually access the register pointed to by the value in the SC\_Address Register. Note that not all addresses are implemented.

---

**Note:** There is no error detection on USC internal addresses.

---

## 4.1.2 USC Internal Register Definitions

The USC internal control and status registers are divided into the following groups:

- USC Overall Control and Status Registers - These registers control overall functions such as chip reset, and provide overall error status
- USC Per UPA Port Registers - These registers contain fields for customizing the port for each device that might be plugged into that slot
- USC Memory Controller Registers - These registers control memory addressing unctions of the system memory

### 4.1.2.1 USC Register Notation Conventions

The fields of each register in the descriptions that follow are given in tabular form in Table 4-6. The left column gives the field name, the second column indicates the bits within the register that the field resides. The third column (Pup-State) specifies the power-up state that the field initializes to after POR, SOFT\_POR, B\_POR, or FATAL. The fourth column is a brief description of the field, and the fifth column specifies a code indicating how the field is accessed. The access codes are described in Table 4-5.

Table 4-5 Register Access Type Codes

Code	Description
R	Read
W	Write
W1C	Write 1 to clear
R0	Read as 0
RW	Read/Write

### 4.1.2.2 USC Register Updating Restrictions

Due to the nature of many of the registers in the USC, special conventions must be followed when updating certain fields within the registers. The conventions below *MUST* be followed at all times in order to ensure proper operation.

- Reads from the USC's internal registers have no side effects and may be done at any time
- SPRQS, SPDQS, SPIQS, and SQUEN in UPA port configuration registers must all be written simultaneously
- Queue sizes must never be decreased. They can only be increased. All queue sizes default to minimum at power-up
- One read should never be cleared until after SPRQS, SPDQS, SPIQS, and SQUEN have been initialized to the desired values. Once this bit is cleared, it cannot be set again except by a reset
- All W1C bits described in Table 4-5 are implemented such that any write to clear which occurs on the same clock cycle as a setting event results in the bit remaining set. The event has precedence over the W1C

### 4.1.3 USC Overall Control Registers

These registers have an overall effect on the USC operation. They are the SC\_Control registers, the SC\_ID register, the SC\_Perf0, SC\_Perf1, and SC\_Perf\_Ctrl registers.

#### 4.1.3.1 SC\_Control Register

The SC\_Control Register is used to implement the USC's reset strategy. It provides three functions: reset cause detection, software reset capability, and wakeup reset control. Table 4-6, "<\$paratext">. shows the detail of the register.

Table 4-6 SC\_Control Register

Field	Bits	Pup-State	Description	Type
POR	31	*[1]	Power-On Reset - Asserted only after the USC sees the assertion of $\overline{\text{SYS\_RESET}}$	R/W1C
SOFT_POR	30	*	Set if the last reset was due to software setting the Soft Power-On Reset bit	R/W
SOFT_XIR	29	*	Set by software to indicate an XIR reset condition	R/W
B_POR	28	*	Set if the last reset was due to the assertion of $\overline{\text{P\_RESET}}$ on the USC. It can be set due to a scan reset, or through a reset generated when writing the frequency margining register	R/W1C
B_XIR	27	*	Set if the last reset was due to the assertion of an $\overline{\text{X\_RESET}}$ on the USC. It can alternatively be set by scan	R/W1C
WAKEUP	26	*	Set if the last reset was due to a wakeup event	R/W1C
FATAL	25	*	Set if a FATAL error was detected by the USC or reported to it. When this bit is set, a system reset will occur (USC CSR bits are not affected)	R/W1C
Reserved	24	0	Reserved	R0
IAP	23	0	Invert parity on the address busses	R/W
EN_WKUP_POR	22	0	Enable wakeup POR generation. Cleared by POR, SOFT_POR, B_POR, FATAL, and WAKEUP	R/W
Reserved	21:0	0	Reserved	R0

1. The highest priority reset source will have its bit set. Only one of the bits marked with an "\*" will be set.

#### 4. Programming Model

Table 4-7 shows the USC reset strategy.

Table 4-7 USC Reset Strategy

Reset Type	Bit Set	Signal Assertions	USC FSMs <sup>[1]</sup>	USC Status Registers
Power on	POR	UPA_RESET[1:0] for 12.8 ms	All reset	All reset
Software	SOFT_POR	UPA_RESET[1:0] for a minimum of 5 ms	All reset	All reset except SOFT_POR
XIR software	SOFT_XIR	$\overline{\text{UPA\_XIR}}$ for one system cycle	Unaffected	Only XIR source affected
Button reset	B_POR	UPA_RESET[1:0] for a minimum of 5 ms	All reset	All reset except B_POR
XIR button	B_XIR	$\overline{\text{UPA\_XIR}}$ for one system cycle	Not affected	Only B_XIR source affected
Wakeup interrupt	WAKEUP	UPA_RESET[1:0] for a minimum of 5 ms	The USC's UPA arbiter is reset	Only WAKEUP source affected
Fatal error condition	FATAL	UPA_RESET[1:0] for a minimum of 5 ms	All reset	Only FATAL source affected

1. FSM = Finite State Machine

Among the reset bits, only one will be set when reset terminates. If multiple resets occur simultaneously, the following order will be chosen for the reporting:

1. POR
2. FATAL
3. B\_POR
4. WAKEUP
5. SOFT\_POR
6. B\_XIR
7. SOFT\_XIR

### *POR - Power-On Reset*

This bit is set if the last reset of the USC was due to the assertion of the  $\overline{\text{SYS\_RESET}}$  pin by an external source, and will occur whenever the machine power cycles.

### *SOFT\_POR - Soft Power-On Reset*

Writing a 1 to this bit has the same effect as power-on reset, except a different status bit in the SC\_Control Register is set. Memory refresh is not affected.

Writing a 0 to this bit clears it and has no other affect.

### *SOFT\_XIR - Soft Externally-Initiated Reset*

Writing a 1 to this bit causes the USC to assert  $\overline{\text{UPA\_XIR}}$  for one cycle. This should cause a software trap for any UPA that is a processor.

### *B\_POR - Button Reset*

This bit is set as a result of a “button” reset which is caused by an external switch and the assertion of  $\overline{\text{P\_BUTTON\_RESET}}$  on the USC. It can also be set by scan and the frequency margining reset. Memory refresh is not affected.

The actions and results of this reset are identical to that of Power-On Reset, except a different status bit is set.

### *B\_XIR - XIR Button Reset*

This bit is set as a result of a “button” XIR reset which is caused by an external switch which asserts the  $\overline{\text{X\_BUTTON\_RESET}}$  on the USC, or by scan.

The actions and results of this reset are identical to that of SOFT\_XIR, except a different status bit is set.

### *WAKEUP - Wakeup*

This bit is set if a wakeup event occurred from one of the UPA ports and EN\_WKUP\_POR is true. A wakeup event is an interrupt to a port that has its “Sleep” bit set - see “SC\_Port\_Config Register Definitions.” Upon a wakeup event, the USC asserts the  $\overline{\text{UPA\_RESET0}}$  to all nodes that can sleep.

Software reads this bit to determine that the last reset it received was due to a wakeup. This allows it to enter the wakeup-related code.

### *FATAL*

This bit is set if one of the following fatal errors was detected by the USC:

1. Address Bus Parity Error (see Section 4.1.4.2, “P0\_ Status Register”)
2. P\_FERR reported by a UPA (see Section 4.1.4.2, “P0\_ Status Register”)
3. A Master Port overflowed its queue (see Section 4.1.4.2, “P0\_ Status Register”)
4. Handshake error between the DPS and Memory Controller (see Section 4.1.5.1, “MC\_Control0 Register”)

Software should also read the registers listed above to determine the actual source of the error and clear any bits associated with it. FATAL is only set if EN\_FATAL is also set.

---

**Note:** In the USC, only cases 1 and 3 will generate a fatal error.

---

### *IAP - Invert Address Parity*

This bit is used by diagnostic software to invert the address parity generated by the USC. This is useful for checking the hardware and software involved in logging address parity errors. It is cleared after any POR and unchanged after XIR or wakeup.

#### **EN\_WKUP\_POR - Enable Wakeup POR**

Enables POR generation on receipt of a wakeup event directed to a port with slave sleep set. Software should set this bit when putting the system to sleep. This bit is cleared on power-up, including a wakeup power-up.

---

**Note:** Interrupts to ports which don't have slave sleep set, including interrupts to invalid ports, will not cause a wakeup reset.

---

### 4.1.3.2 SC\_ID Register

The SC\_ID Register contains read-only ID information for the USC.

Table 4-8 SC\_ID Register

Field	Bits	PupState	Description	Type
JEDEC	31:16	0x3340	USC's JEDEC ID; ID = 0x3340 for the USC	R
UPANUM	15:12	0x3,4 <sup>[1]</sup>	The number of UPA ports supported by this USC implementation	R
Reserved	11:8	0	Reserved	R0
IMPLNUM	7:4	0	Implementation number	R
VERNUM	3:0	00	Version number; starts at 0 and increments for each revision of this ASIC	R

1. 3 for the SBus reference platform

### 4.1.3.3 SC\_Perf0 Register

The SC\_Perf0 Register is a 32-bit read-only register. Value read back contains the counts of the event selected by the SC\_Perf\_Ctrl register. Whenever this register is read, the shadow register is updated with the contents of the SC\_Perf1 counter. When the counter reaches its maximum count, it will wrap around to 0x0 and continue to count. Software needs to detect and handle the overflow condition.

Table 4-9 SC\_Perf0 Register

Field	Bits	Description	Type
CNT0[31:0]	31:00	Contains a value for event counter 0	R



#### 4.1.3.4 SC\_Perf1 Register

The SC\_Perf1 Register is a 32-bit read-only register. Value read back contains the counts of the event selected by the SC\_Perf\_Ctrl register. Whenever this register is read, the shadow register is updated with the contents of the SC\_Perf0 counter. 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 4-10 SC\_Perf1 Register

Field	Bits	Description	Type
CNT1[31:0]	31:00	Contains a value for event counter 1	R

#### 4.1.3.5 SC\_PerfShadow Register

The SC\_PerfShadow Register is a 32-bit read-only register. Value read back contains the count of the event shadowed by the read of the last SC\_Perf register. The shadow provides a mechanism to “atomically” read the values of both counters.

The shadow register is sampled at the time byte 0 is read for either performance register.

Table 4-11 SC\_PerfShadow Register

Field	Bits	Description	Type
CNT1[31:0]	31:00	Contains a value for event counter 0 or 1	R

#### 4.1.3.6 SC\_Perf\_Ctrl Register

The SC\_Perf\_Ctrl Register controls the events to be monitored by the Performance Counter Registers. The event counter in the Performance Counter Register will be reset when the CLR bit is asserted. Sources for each of the counters is given in Table 4-12, <"\$paratext">. and Table 4-13, <"\$paratext">. of this section.

---

**Note:** There are only two counters with each counter multiplexing one of 16 possible sources. It is not possible to track more than two events simultaneously.

---

**Table 4-12** Performance Monitor Control Register

Field	Bits	Description	Type
Reserved	31:16	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. Counter 1 is cleared when the CLR1 field is written. It resets to 0xf. This value was chosen to minimize power.	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. Counter 0 is cleared when the CLR0 field is written. It resets to 0xf. This value was chosen to minimize power.	R/W

The performance registers each measure one of 16 total events. A list of possible selections for counter 0 is shown in Table 4-13.

**Table 4-13** USC Performance Counter 0 Event Sources

SEL	Event Sources
0x0	System clock count
0x1	Number of PRequests from all sources
0x2	Number of PRequests from Processor 0
0x4	Number of PRequests from U2S
0x5	Number of cycles the UPA 128-bit data bus is busy
0x6	Number of cycles the UPA 64-bit data bus is busy
0x7	Number of cycles stalled during PIO
0x8	Number of memory requests issued
0x9	Number of cycles Memory Controller is busy <sup>[1]</sup>
0xA	Number of cycles stalled because of a pending transaction scoreboard hit
0xB	Number of coherent write miss requests from P0 (RDO)
0xC	Number of coherent write miss requests from P1 (RDO)

#### 4. Programming Model

Table 4-13 USC Performance Counter 0 Event Sources

SEL	Event Sources
0xD	Number of coherent intervention transactions
0xE	Number of data transactions from the U2S
0xF	Number of coherent read transactions issued

1. This includes all events which cause the Memory Controller to be non-idle including memory references and refresh.

Similarly, Counter 1 counts one of sixteen possible sources. A list of possible selections is shown in Table 4-13. Note that this list contains some duplications with the Counter 0 event sources *and* some unique items. Common sources retain the same SEL number.

Table 4-14 USC Performance Counter 1 Event Sources

SEL	Event Sources
0x0	System clock count
0x1	Number of PRequests from all sources
0x2	Number of PRequests from Processor 0
0x4	Number of PRequests from the U2S
0x5	Read requests from P0
0x6	Coherent read misses from P0
0x7	Number of PIO accesses (NCxx) from P0
0x8	Number of memory requests issued
0x9	Number of memory requests completed
0xA	Read requests from P1
0xB	Coherent read misses from P1
0xC	Number of PIO accesses (NCxx) from P1
0xD	Number of coherent write transactions issued
0xE	Number of data transactions from the U2S
0xF	Reserved

#### 4.1.4 USC Port Interface Registers

There are multiple copies of these registers (as specified). They are used to enable and disable certain port features. Each of the registers is listed individually to indicate which bits are hardwired in this implementation. Those bits which are hardwired (read only) are shown with the normal field name rather than being listed as reserved.

##### 4.1.4.1 P0\_Config Register

Table 4-15 shows the contents of Port 0 (Processor 0) configuration register.

Table 4-15 P0\_Config Register

Field	Bits	PupState	Description	Type
MD <sup>[1]</sup>	31	0	Master Disable. If set, the USC will block all requests in its master request queue for this master port. All processor ports power-up enabled. Once enabled, any requests which exist in the master request queues will proceed.	R/W <sup>[1]</sup>
S_Sleep	30	0	Slave Sleep. If set, the slave is currently asleep and an interrupt packet directed to that slave results in a wakeup sequence by the USC and the interrupting port receives a NACK reply. If clear, interrupt packets are treated normally.	R/W
Reserved	29:28	0	Reserved	R0
SPRQS <sup>[2]</sup>	27:24	1	Slave PRequest Queue Size. Software initializes this field with the size in 2-cycle packets of the corresponding slave request queue.	R/W
Reserved	23:18	0	For ports implementing this feature, this would be the slave port data queue size. It is not programmable in this implementation.	R
SPIQS	17:16	0	Slave Port Interrupt Queue Size. Software programs this field with the length in address packets of the corresponding interrupt queue.	R/W
SQUEN	15	0 (write only)	Writes of SPRQS and SPIQS are qualified with this bit. If set, the values in these fields are updated. If not, they are unchanged.	W
Oneread	14	1	Software sets to 0 if the number of outstanding reads allowed to this port is greater than 1. When set to 1, the USC will accept P_RASB, P_RAB, or P_RAS replies. When cleared, only P_RAB or P_RAS are valid.	R/W
Reserved	13:0	0	Read as 0. Not writable.	R

1. In the reference platform, this bit is read only and returns 0 (i.e., the port is always enabled). It does not make sense to disable the only processor port.

2. An implementation may choose to set a maximum value which is less than what can be programmed in this register. In that case, a larger value must still result in correct operation, but not the desired number of requests being forwarded.

#### 4.1.4.2 P0\_Status Register

Table 4-16 is the status register of Port 0.

Table 4-16 P0\_Status Register

Field	Bits	PupState	Description	Type
FATAL <sup>[1]</sup>	31	0	This bit is set if this port sent a P_FERR error reply or if a port returns a P_RASB reply when oneread is FALSE. This condition will cause a chip reset.	R/W1C
IADDR	30	0	This bit is set if this port attempted to send a request (read OR write) to an illegal address or a nonexistent port. This is an advisory bit.	R/W1C
IPORT	29	0	This bit is set if this port attempted to send an interrupt packet to an illegal destination port, or that port's SPIQS field is set to 0. This is an advisory bit and the interrupt packet is dropped (normal ACK).	R/W1C
IPRTY	28	0	This bit is set if a packet sent from this port resulted in the USC detecting an address parity error. It has priority over any other fatal condition (e.g., if IPRTY, you may also see IADDR; software must sort out this case). This condition will cause a chip reset.	R/W1C
MC0OF	27	0	Master Class 0 Over Flow. This bit is set if this master tried to send a packet when no space was available in the queue. This condition causes a chip reset.	R/W1C
MC1OF	26	0	Master Class 1 Overflow. This bit is set if this master tried to send a packet when no space was available in the queue. This condition causes a chip reset.	R/W1C
MC0Q	25:23	See Table 4-19	These bits indicate to the system software the number of request packets that can be issued to the master class 0 before it overflows.	R
MC1Q	22:20	See Table 4-19	These bits indicate to the system software the number of request packets that can be issued to the master class 1 before its queue overflows.	R
Reserved	19:0	0	Read only as 0	R

1. Refer to the FATAL bit in the section entitled "SC\_Control Register."

### 4.1.4.3 SYSIO\_Config Register

Table 4-17 shows the contents of the UPA-to-SBus Interface configuration register.

Table 4-17 SYSIO\_Config Register

Field	Bits	PupState	Description	Type
MD	31	1	Master Disable. If set, the USC will block all requests in its master request queue for this master port. Once enabled, any requests which exist in the master request queues will proceed. This bit must be set to 0 for DVMA or interrupts.	R/W
Reserved	30:28	0	Reserved; would be slave sleep for a port implementing this bit.	R
SPRQS <sup>[1]</sup>	27:24	1	Slave PRequest Queue Size. Software initializes this field with the size in 2-cycle packets of the corresponding slave request queue.	R/W
SPDQS	23:18	4	Slave Port Data Queue Size. Software initializes this field with the length in address packets of the corresponding slave data queue. <i>The UPA specification dictates that this field be present, but hardware ignores any nonzero value. If SPDQS is nonzero, then hardware assumes its value to be four times SPRQS.</i>	R/W
Reserved	17:16	0	Reserved. It would be Slave Port Interrupt Queue Size for devices implementing this feature.	R
SQUEN	15	0 (write only)	Software sets this bit when updating SPRQS, SPDQS, and SPIQS, which must be done all at once.	W
Oneread	14	1	The U2S may be set as a oneread, or multi-read device. Value should be programmed after a probe of the UPA port ID register for the U2S.	R/W
Reserved	13:0	0	Reserved	R

1. An implementation may choose to set a maximum value which is less than what can be programmed in this register. In that case, a larger value must still result in correct operation, but not the desired number of requests being forwarded.

#### 4.1.4.4 SYSIO\_Status Register

Table 4-18 is the status register of the UPA-to-SBus interface.

Table 4-18 SYSIO\_Status Register

Field	Bits	PupState	Description	Type
FATAL <sup>[1]</sup>	31	0	This bit is set if this port sent a P_FERR error reply or a P_RASB reply when oneread is false. This condition will cause a chip reset.	R/W1C
IADDR	30	0	This bit is set if this port attempted to send a request to an illegal address or a non-existent port. This is an advisory bit.	R/W1C
IPORT	29	0	This bit is set if this port attempted to send an interrupt packet to an illegal destination port. This is an advisory bit.	R/W1C
IPRTY	28	0	This bit is set if a packet sent from this port resulted in the USC detecting an address parity error. This condition can cause a chip reset.	R/W1C
MC0OF	27	0	Master Class 0 Over Flow. This bit is set if this master tried to send a packet when no space was available in the queue. This condition will cause a chip reset.	R/W1C
Reserved	26	0	Reserved. This bit would be Master Class 1 Over Flow for ports implementing this feature.	R
MC0Q	25:23	See Table 4-19	These bits indicate to system software the number of requests packets that can be issued to master class 0 before it overflows	R
Reserved	22:20	0	Reserved. Would indicate Master Class 1 request packets for ports implementing this feature.	R
Reserved	19:0	0	Reserved	R

1. Refer to the FATAL bit in the section entitled "SC\_Control Register."

The port number and the initial values of MCOQ and MCIQ (specified in Table 4-16 and Table 4-18) are given in Table 4-19.

Table 4-19 SC Port\_Status Register MC0Q and MC1Q Init Table

Port/Class	Type of Master	Queue Size
Port 0, Class 0	Processor 0	1
Port 0, Class 1	Processor 0	4
Port 1, Class 0	Processor 1 <sup>[1]</sup>	1
Port 1, Class 1	Processor 1 <sup>[1]</sup>	4
Port 2, Class 0	U2S <sup>[2]</sup>	2

1. Only with the MP System

2. Only one class is permitted for the U2S

## 4.1.5 Memory Controller Registers

The registers described below control the functionality of the USC Memory Controller.

### 4.1.5.1 MC\_Control0 Register

The MC\_Control0 Register is also referred to as Memory Control Register 0. It contains a number of bits which program the Refresh Controller in the Memory Controller.

Table 4-20 MC\_Control0 Register

Field	Bits	PupState	Description	Type
RefEnable	31	0	Refresh enable	R/W
Reserved	30:16	0	Reserved. Read as 0	R0
SIMMPresent[7:0]	15:8	0x0F	Determines which SIMMs to refresh	R/W
RefInterval[7:0]	7:0	0x00	Interval between refreshes. Each encoding is eight system clocks	R/W

#### RefEnable

Main memory is composed of dynamic RAMs which require periodic “refreshing” in order to maintain the contents of the memory cells. The RefEnable bit is used to enable the refresh of main memory.

0 = disable refresh

1 = enable refresh

Disabling refresh causes the RefInterval timer to stop counting.

POR is the only condition to clear RefEnable.

SOFT\_POR, B\_POR, FATAL, WAKEUP, B\_XIR, and SOFT\_XIR leave RefEnable unchanged.

---

**Note:** This bit may be written to at any time. Any refresh operation in progress at the time of clearing will be aborted with a possible loss of data.

---



*SIMMPresent[7:0]*

SIMMPresent[7:0] is used to indicate the presence/absence of SIMMS so that the USC does not spend more time than is necessary refreshing unpopulated SIMMs. A 0 in this field indicates the corresponding SIMM is not present, a 1 indicates presence. These bits must be set by software after probing.

The SIMM-to-bit correspondence is given in Table 4-21.

Table 4-21 SIMMPresent Encoding

SIMMPresent<i>	SIMM Slot
0	0
1	1
2	2
3	3

---

**Note:** Refresh must be disabled first by clearing the RefEnable bit before changing this field, or the RefInterval. Refresh may be enabled again simultaneously with writing SIMMPresent and RefInterval. Failure to follow this rule may result in unpredictable behavior.

---

*RefInterval*

RefInterval specifies the interval time between refreshes, in quanta of eight system cycles. Software should program RefInterval according to Table 4-22. Values given are in decimal and are derived from the following formula:

$$refValue = \frac{refreshPeriod}{numberOfRows \times UPAClockPeriod \times 8 \times numberOfPairs}$$

Table 4-22 RefInterval Table for SIMMs Using 4 MB (1024 Rows/16 ms), 16 MB (4096 Rows/64 ms), or 64 MB (8192 Rows/128 ms)

SIMM pairs	UPA Cycle Time						
	10ns	12 ns	14 ns	15 ns	16 ns	18 ns	20 ns
1	195	162	139	130	121	108	97
2	98	81	69	65	60	54	48
3	65	54	46	43	40	36	32
4	49	40	34	32	30	27	24

---

**Note:** A value of 0 corresponds to refresh disabled. Values of 1 and 2 are illegal and should not be used. They are not checked in hardware.

---

#### 4.1.5.2 MC\_Control1 Register

Memory Control Register 1, Table 4-23, contains fields which control the read, write, and refresh timing for the DRAM SIMMs. They allow software to optimize memory access timing for a particular system frequency.

The contents of Memory Control Register 1 may be changed as required by any electrical tuning of memory timing based on detailed Spice analysis. See Table 4-24, <"\$paratext">. for the proper programming values for this register.

---

**Note:** Only 60-ns DRAMs are supported in the reference platform.

---

Table 4-23 MC\_Control Register 1

Field	Bits	PupState	Description	Type
Reserved	31:13	0	Reserved read as 0	R0
CSR	12	1	$\overline{\text{CAS}}$ -to- $\overline{\text{RAS}}$ delay for CBR refresh cycles	R/W
WPC1	11:10	11	Page cycle 1 write	R/W
RCD	9	1	$\overline{\text{RAS}}$ -to- $\overline{\text{CAS}}$ delay	R/W
CP	8	1	$\overline{\text{CAS}}$ precharge	R/W
RP	7:6	11	$\overline{\text{RAS}}$ precharge	R/W
RAS	5:4	11	Length of $\overline{\text{RAS}}$ for precharge	R/W
PC0	3:2	11	Page cycle 0	R/W
PC1	1:0	11	Page cycle 1	R/W

##### CSR - $\overline{\text{CAS}}$ -Before- $\overline{\text{RAS}}$ Delay Timing

Controls the  $\overline{\text{CAS}}$  assertion to  $\overline{\text{RAS}}$  assertion delay for  $\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$  (CBR) refresh cycles.

0 = one clock between  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$

1 = two clocks between  $\overline{\text{CAS}}$  and  $\overline{\text{RAS}}$

#### 4. Programming Model

##### *WPC1 - Page Cycle 1 Write Timing*

Controls the  $\overline{\text{RAS}}$  and  $\overline{\text{CAS}}$  low time during writes of the second block of data.

00 =  $\overline{\text{RAS}}$  and  $\overline{\text{CAS}}$  low for two system clocks

01 =  $\overline{\text{RAS}}$  and  $\overline{\text{CAS}}$  low for three system clocks

10 = reserved

11 = reserved

##### *RCD - $\overline{\text{RAS}}$ -to- $\overline{\text{CAS}}$ Delay*

RCD controls the  $\overline{\text{RAS}}$ -to- $\overline{\text{CAS}}$  delay during the initial part of the read or write memory cycle.

0 = two system clocks between the assertion of  $\overline{\text{RAS}}$  and the assertion of  $\overline{\text{CAS}}$

1 = three system clocks between the assertion of  $\overline{\text{RAS}}$  and the assertion of  $\overline{\text{CAS}}$

##### *CP - $\overline{\text{CAS}}$ Precharge*

CP controls the  $\overline{\text{CAS}}$  precharge time in between page cycles.

0 = one clock of  $\overline{\text{CAS}}$  precharge

1 = two clocks of  $\overline{\text{CAS}}$  precharge

##### *RP - $\overline{\text{RAS}}$ Precharge*

RP controls the  $\overline{\text{RAS}}$  precharge time during reads and writes.

00 = three system clocks of  $\overline{\text{RAS}}$  precharge

01 = four system clocks of  $\overline{\text{RAS}}$  precharge

10 = five system clocks of  $\overline{\text{RAS}}$  precharge

11 = Reserved

## $\overline{RAS}$

$\overline{RAS}$  is used to control the length of time that  $\overline{RAS}$  is asserted during refresh cycles.

00 =  $\overline{RAS}$  asserted for four clocks

01 =  $\overline{RAS}$  asserted for five clocks

10 =  $\overline{RAS}$  asserted for six clocks

11 = Reserved

## PC0 - Page Cycle 0

PC0 controls the time to deassertion of  $\overline{CAS}$  for the read of the first half of the memory block. It does not affect the timing for writes, which is fixed at four clocks.

00 =  $\overline{CAS}$  assertion for two clocks

01 =  $\overline{CAS}$  assertion for three clocks

10 =  $\overline{CAS}$  assertion for four clocks

11 =  $\overline{CAS}$  assertion for five clocks

## PC1 - Page Cycle1

PC1 controls the timing of the  $\overline{RAS}$  and  $\overline{CAS}$  for the read of the second half of the memory block, see Figure 4-1, "PC1 Field Timing Effects."

PCI	Relative Timing
00	$\overline{RAS}$ L L L H $\overline{CAS}$ H L L H
01	$\overline{RAS}$ L L L L H $\overline{CAS}$ H L L L H
10	$\overline{RAS}$ L L L L H H $\overline{CAS}$ H L L L L H
11	$\overline{RAS}$ L L L L H H $\overline{CAS}$ L L L L L H

Figure 4-1 PC1 Field Timing Effects

#### 4. Programming Model

The values of this control register as a function of the system operating frequency are given in Table 4-24.

Table 4-24 MC\_Control1 Values as a Function of Operating Frequency

Clock Period (ns)	CSR	WPC 1	RCD	CP	RP	$\overline{\text{RAS}}$	PC0	PC1
12	0	00	1	0	01	10	10	10
14	0	00	1	0	00	01	01	01
15	0	00	0	0	00	01	10	01
16	0	00	0	0	00	00	10	01
18	0	00	0	0	00	00	01	01
20	0	00	0	0	00	00	01	00

Initialization of the MC\_Control registers should be performed in accordance with the probing algorithm described in related documents.

---

**Note:** The fields of this register must be initialized before *any* memory operation, including refresh, can begin. Should software wish to change any of these values, it must do so by first inhibiting all memory references. Then refresh must be disabled. The control processor must wait a few dozen clocks after disabling refresh to ensure that any pending refresh operation has completed. Accesses on the EBus may be sufficiently slow to satisfy this requirement. New values may now be programmed.

---



## 5.1 Introduction

One of the major functions of the USC is the forwarding and handling of UPA packets. This chapter describes in detail how the USC accomplishes this.

## 5.2 USC Transaction Table

Table 5-1, “USC Transaction Table,” shows all the possible transactions that the USC will handle and all possible outcomes.

Some notes about the table:

1. MEM\_ADDR is an address in the main memory address space. It is the space defined by PA[40] = 0.
2. SLV\_ADDR is an address in the slave address space of a UPA port that is present in the system. This includes CPU\_ADDR, SYSIO\_ADDR, and FFB\_ADDR (if FFB is present).
3. CPU\_ADDR is an address in the address space located in the CPU.
4. SYSIO\_ADDR is an address in the address space located in the U2S.
5. FFB\_ADDR is an address in the address space located in the FFB, if FFB is present. If FFB is not present, then the FFB address space becomes ERR\_ADDR.
6. ERR\_ADDR is an address that is not supported in the system. It is the total 41-bit address space excluding [MEM\_ADDR + CPU\_ADDR + SYSIO\_ADDR + FFB\_ADDR (if not present)].
7. VAL\_ID is a valid interrupt target ID. In the uniprocessor reference platform, the CPU is the only valid target ID.

8. INV\_ID is an invalid interrupt target ID.
9. IADDR refers to the IADDR bit in the SC\_Port\_Status register of the initiator.
10. IPORT refers to the IPORT bit in the SC\_Port\_Status register of the initiator.

Table 5-1 USC Transaction Table

Transaction Type	Master ID	Address	PRequest/ SRequest	P_Reply	S_Reply to Master	S_Reply to Slave
P_NCRD_REQ	CPU, U2S	CPU_ADDR, SYSIO_ADDR (OneRead=0)	P_NCRD_REQ	P_RAS P_RERR P_RTO	S_RAS S_ERR S_RTO	S_SRS – –
		CPU_ADDR, SYSIO_ADDR (OneRead=1)	P_NCRD_REQ	P_RASB P_RERR P_RTO	S_RAS S_ERR S_RTO	S_SRS – –
		FFB_ADDR	P_NCRD_REQ	P_RASB	S_RAS	S_SRS
		MEM_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
P_NCBRD_REQ	CPU, U2S	CPU_ADDR, SYSIO_ADDR (OneRead=0)	P_NCBRD_REQ	P_RAB P_RERR P_RTO	S_RBU S_ERR S_RTO	S_SRB – –
		CPU_ADDR, SYSIO_ADDR (OneRead=1)	P_NCBRD_REQ	P_RASB P_RERR P_RTO	S_RBU S_ERR S_RTO	S_SRB – –
		FFB_ADDR	P_NCBRD_REQ	P_RASB	S_RBU	S_SRB
		MEM_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
P_NCWR_REQ	CPU, U2S	CPU_ADDR	_ [1]	–	S_WAS	(Data dropped)
		SYSIO_ADDR, FFB_ADDR	P_NCWR_REQ	P_WAS	S_WAS	S_SWS
		MEM_ADDR	(set IADDR)	–	S_WAS	(Data dropped)
		ERR_ADDR	(set IADDR)	–	S_WAS	(Data dropped)
P_NCBWR_REQ	CPU, U2S	CPU_ADDR	_ [1]	–	S_WAS	(Data dropped)
		SYSIO_ADDR, FFB_ADDR	P_NCBWR_REQ	P_WAB	S_WAB	S_SWB
		MEM_ADDR	(set IADDR)	–	S_WAB	(Data dropped)
		ERR_ADDR	(set IADDR)	–	S_WAB	(Data dropped)



## 5. Packet Handling

Table 5-1 USC Transaction Table

Transaction Type	Master ID	Address	PRequest/ SRequest	P_Reply	S_Reply to Master	S_Reply to Slave
P_INT_REQ  (interrupt nacked)	CPU, U2S	VAL_ID	P_INT_REQ	P_IACK	S_WAB	S_SWIB
		INV_ID	(set IPORT)	–	S_WAB	(Data dropped)
		VAL_ID	–	–	S_INAK	
P_RDS_REQ	CPU	MEM_ADDR	–	–	S_RBU	(from mem)
		SLV_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
	U2S	Any address	Undefined <sup>[2]</sup>	–	–	–
P_RDSA_REQ	CPU	MEM_ADDR	–	–	S_RBS	(from mem)
		SLV_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
	U2S	Any address	Undefined <sup>[2]</sup>	–	–	–
P_RDO_REQ	CPU	MEM_ADDR	–	–	S_RBU	(from mem)
	U2S	MEM_ADDR	S_CPI_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_RBU S_RBU S_RBU	S_CRAB S_CRAB (from mem)
	CPU, U2S	SLV_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
P_RDD_REQ	CPU	MEM_ADDR	–	–	S_RBS	(from mem)
	U2S	MEM_ADDR	S_CPD_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_RBS S_RBS S_RBS	S_CRAB S_CRAB (from mem)
	CPU, U2S	SLV_ADDR	(set IADDR)	–	S_ERR	–
		ERR_ADDR	(set IADDR)	–	S_RTO	–
P_WRB_REQ (Data has been invalidated)	CPU, U2S	MEM_ADDR	–	–	S_WAB	(to mem)
	CPU	MEM_ADDR	–	–	S_WBCAN	–
	CPU, U2S	SLV_ADDR	Undefined <sup>[3]</sup>	–	–	–
		ERR_ADDR	Undefined <sup>[4]</sup>	–	–	–
P_WRI_REQ (IVA set)	CPU	MEM_ADDR	S_INV_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_WAB S_WAB S_WAB	(to mem) (to mem) (to mem)
P_WRI_REQ (IVA not set)	CPU	MEM_ADDR	–	–	S_WAB	(to mem)

Table 5-1 USC Transaction Table

Transaction Type	Master ID	Address	PRequest/ SRequest	P_Reply	S_Reply to Master	S_Reply to Slave
P_WRI_REQ (IVA = X)	U2S	MEM_ADDR	S_INV_REQ (to CPU)	P_SACK P_SACKD P_SNACK	S_WAB S_WAB S_WAB	(to mem) (to mem) (to mem)
	CPU, U2S	SLV_ADDR	(set IADDR)	–	S_WAB	(Data dropped)
		ERR_ADDR	(set IADDR)	–	S_WAB	(Data dropped)

1. Any P\_NCWR\_REQ/P\_NCBWR\_REQ transactions directed to the processor are not forwarded (because the processor has a slave data queue size of 0), and data is dropped on the floor.
2. The U2S is incapable of generating a P\_RDS\_REQ or P\_RDSA\_REQ transaction. If the U2S does generate one of these transactions, the USC will not detect it as an error, and the results are undefined.
3. The USC does not support cacheable address space at the UPA slaves. If a UPA port issues a P\_WRB\_REQ to a UPA slave, then a hardware error has occurred since it could not have ever successfully completed a coherent read to that address. However, the USC will not detect it as an error and the results are undefined.
4. If a UPA port issues a P\_WRB\_REQ to an illegal or invalid address, then a hardware error has occurred since it could not have ever successfully completed a coherent read to that address. However, the USC will not detect it as an error and the results are undefined.

## 5.3 Coherent Transactions

### 5.3.1 Coherence Algorithm

Coherent transactions (P\_RDS\_REQ, P\_RDSA\_REQ, P\_RDD\_REQ, P\_RDO\_REQ, P\_WRB\_REQ, and P\_WRI\_REQ) can only be issued to main memory. The USC does not support cacheable accesses to any slave address space other than main memory. Coherent requests always result in the transfer of an entire block of data (64 bytes).

The USC implements a no data tag system. It's designed to work with a U2S which has only a single-line merge buffer. There are very strict rules on how the U2S handles this merge buffer.

The PIF maintains a three-entry coherence scoreboard, with one entry for each source: processor class 0, processor class 1, and U2S. There can be only one coherent transaction outstanding from each source at any one time. Whenever the PIF wants to execute a coherent transaction, it will check the scoreboard to see if there is a matching index or a pending read-modify-write (RMW) from the U2S. Index checking is based on a PA[18:6] (these bits were chosen because they are the index bits for a 512 MB cache).

The U2S maintains a single 64-byte merge buffer for performing writes smaller than 64 bytes to coherent space. If the U2S wants to perform a write of less than 64 bytes to coherent space, then it must issue a P\_RDO\_REQ, perform the write, and flush it back to main memory immediately using P\_WRB\_REQ. If the U2S merely wants to read a block with no intent to write, then it issues a P\_RDD\_REQ transaction. If the U2S want to do a block write to memory, then it uses P\_WRI\_REQ.

When the U2S has possession of a memory block, any requests to that block from the processor will block until the U2S writes it back to main memory. The PIF will maintain the address in the scoreboard and set the RMW pending flag. If another memory request hits on that address, then that request and all subsequent requests from that class will stall. After the U2S issues the P\_WRB\_REQ, then the RMW flag will be cleared and stalled transactions will unblock. Therefore, it is important that the U2S write back the block as soon as possible.

The USC will never issue an SRequest (S\_INV\_REQ, S\_CPB\_REQ, S\_CPI\_REQ, or S\_CPD\_REQ) to the U2S because of a coherent request issued from the processor. The IVA bit in P\_WRI\_REQ issued from the U2S is ignored; no S\_INV\_REQ is sent to the U2S.

Every time the U2S issues a coherent read transaction, the USC must then send an SRequest to the processor to cause a copy back and/or invalidation.

The U2S is not allowed to issue P\_RDS\_REQ or P\_RDSA\_REQ.

The USC will not detect the case when the U2S does not issue a P\_WRB\_REQ to flush the block from the merge buffer back to main memory (for example, two back-to-back P\_RDO\_REQs). This is a coherence error and what happens is undefined. Also, the USC will not check the addresses in the P\_RDO\_REQ and P\_WRB\_REQ packets to make sure that they match.

### 5.3.2 Cancelled Writebacks

The PIF also maintains a “cancel” bit. The cancel bit is initially cleared on reset. If the processor responds with a P\_SACKD P\_Reply to a S\_CPI\_REQ or S\_INV\_REQ request from the USC (meaning that the block is in the writeback buffer), then the cancel bit is set. When the USC sees the corresponding P\_WRB\_REQ from the processor, then it issues an S\_WBCAN to the processor, and clears the cancel bit.

Once the USC sees a P\_SACKD response, it keeps track of the address of the block which caused the cancel bit to be set. (PA[29:6] is used, since the USC only supports up to 1 GB of memory.)

If the USC needs to send an S\_CPD\_REQ/S\_CPI\_REQ/S\_INV\_REQ to the processor, it will do an address compare. If the address matches, the USC knows that the block has already been invalidated and will go straight to memory to get the block, without sending any SRequest to the UltraSPARC-I processor. If the address does not match, it will send the SRequest to the processor without waiting for the cancel flag to be cleared.

The USC assumes that from the processor, no concurrent P\_WRB\_REQ and P\_WRI\_REQ are externally visible in any way. In particular, if the processor issues a P\_WRI\_REQ with the IVA bit set, then when the USC sends the processor an S\_INV\_REQ, it should never receive a P\_SACKD reply.

In general, when the UltraSPARC-I processor issues a P\_WRI\_REQ to the USC, and the S\_Reply for the WRI is not received yet, and if there is a dirty block that needs to be written back to memory, then any S\_REQ sent to the processor should not get a P\_SACKD reply.

Conversely, between an UltraSPARC-I processor P\_WRB\_REQ and the corresponding S\_Reply, a pending P\_WRI\_REQ inside the UltraSPARC-I processor will not be observed at the UPA interface.

This is documented as a requirement of the UltraSPARC-I processor in the UPA errata document.

### *5.3.3 Request Flow*

#### *5.3.3.1 Coherent Read from Processor*

A coherent read from the processor is received by the PIF and forwarded to the MC if the MC is not busy processing a previous memory request. The MC maintains a single-entry memory-request queue. If the MC is busy performing a refresh, then the memory request is queued. The PIF will not forward the read packet if the U2S is currently performing an RMW on it, or if there is another active transaction with the same index.

When the MC receives the packet, it begins a memory read transaction immediately. It also makes a request to the DPS, telling the DPS that a certain master wants to do a read. The MC then proceeds to load the XB1 (crossbar switch) read buffer with the data. The MC uses a handshake to inform the DPS when the first and second halves of the block have been loaded into the XB1 read buffer. The DPS will issue the S\_Reply to the initiator and drain the read buffer whenever there is sufficient data to hand over, using UPA\_DATA\_STALL if necessary.

The MC will not issue another read request to the DPS unless the DPS has informed it that it has completely drained the read buffer. This prevents read data from being overwritten. In this way, filling the read buffer from memory and draining the read buffer and decoupling it if there is a lot of other port-to-port activity on the UPA data buses. On the other hand, if the UPA data buses are idle, then data can be delivered immediately as quickly as memory can supply it.

### 5.3.3.2 Coherent Write From Processor

Coherent writes from the processor are handled similarly to coherent reads, with the following differences.

When the MC receives a coherent write from the PIF, it will issue a data path request immediately to the DPS. The DPS will immediately issue an S\_Reply to the initiator to begin sourcing data, and the DPS sets up the XB1 write buffer to begin receiving data. As soon as the write buffer is filled with 288 bits of data, then the DPS signals the MC, and the MC writes the first half of the memory block into memory. The MC will wait for four clocks, then it will write the second half of the memory block into memory. UPA\_DATA\_STALL is never asserted for writes, and four clocks is the worst-case time for delivery of the second half of the block on the UPA data bus (from the U2S). The MC will not issue another read or write request until the write is completely finished.

### 5.3.3.3 Coherent Read from the U2S

The U2S is only allowed to issue P\_RDO\_REQ or P\_RDD\_REQ. Since it has no cache, it cannot issue P\_RDS\_REQ or P\_RDSA\_REQ.

When the U2S issues a coherent read from the U2S, the chain of events is very similar to coherent reads issued from the processor. In this case, however, the USC is required to issue an SRequest to the processor (S\_CPI\_REQ/S\_CPD\_REQ). Depending on whether a P\_SACK/P\_SACKD or P\_SNACK reply is issued from the processor, the USC will source the data either from the processor's cache or memory, respectively.

The issuing of the SRequest to the processor and the issuing of the memory request to the MC are coupled together. That is, both resources must be free before the transaction can proceed; the issuance of the SRequest and memory request cannot be decoupled.

The MC will always perform the memory read operation, filling the XB1 read buffer with data from memory. However, the DPS must wait for the P\_Reply to determine from where to source the data. If the DPS determines that it has to source the data from the processor's cache, then the data in the XB1 is dropped on the floor and thrown away.

Because it takes a long time for the USC to send the SRequest, and for the processor to send back the P\_Reply, the latency for coherent reads from main memory issued by the U2S is longer than coherent reads issued from the processor.

#### *5.3.3.4 Coherent Write from the U2S*

The U2S will issue P\_WRB\_REQ (as part of a RMW) or P\_WRI\_REQ to inject new data into the coherence domain. If the USC sees an P\_WRB\_REQ, it assumes that it is coupled with a previous P\_RDO\_REQ. The write is issued to main memory, and the memory block becomes "unblocked;" that is, the processor can now access this memory block.

If the U2S sees an P\_WRI\_REQ, it issues an S\_INV\_REQ to the processor, then waits for the P\_Reply to come back before performing the write to memory.

#### *5.3.4 Coherent Requests to the FFB*

Transactions to the FFB require special handling because FFB does not generate a full 5-bit P\_Reply. It is unable to signal an error to the USC. If a coherent request is issued to the FFB, it has no way of signalling P\_RERR. To handle these transactions, the USC will generate an S\_ERR on behalf of the FFB if it sees an P\_RDS\_REQ, P\_RDSA\_REQ, P\_RDD\_REQ, or P\_RDO\_REQ directed to the FFB. The packet will not be forwarded to the FFB, it will be intercepted by the USC. If the USC sees P\_WRB\_REQ issued to the FFB, the results are undefined. An P\_WRI\_REQ is not forwarded and the write data is dropped on the floor.

### *5.4 Noncached Transactions*

Noncached transactions (P\_NCBWR\_REQ, P\_NCWR\_REQ, P\_NCBRD\_REQ, and P\_NCRD\_REQ) can be generated by the processor or the U2S. They are simpler than coherent transactions because handling of these transactions is limited to the PIF and DPS, and the MC is never involved.

### 5.4.1 Noncached Reads

For noncached reads (P\_NCBRD\_REQ and P\_NCRD\_REQ), the PIF will forward the packet to the appropriate slave, and wait for an P\_Reply from the slave. Then the DPS will issue the S\_Replies to both master and slave to transfer the data. If the P\_Reply indicates an error, then this is forwarded to the master by the DPS.

Noncached reads are only allowed to go to slave address space. If an NC read is directed to cached address space, the USC will issue an S\_ERR.

### 5.4.2 Noncached Writes

For noncached writes (P\_NCBWR\_REQ and P\_NCWR\_REQ), the PIF will forward the packet to the appropriate slave. The DPS will issue the S\_Replies to master and slave to transfer the data. Then the PIF waits for an P\_Reply to be issued from the slave, so that it can decrement its count of outstanding transactions to that slave.

Noncached writes are only allowed to go to slave address space. If an NC write is directed to cached address space, the USC will drop the data on the floor.

Noncached writes are not forwarded to the processor, because the PREQ\_DQ[5:0] field in the UltraSPARC-I processor's UPA\_PORT\_ID register is 0, and therefore the UltraSPARC-I processor cannot accept any write data. In this case, the USC will drop the data on the floor. This behavior is hardwired into the USC and cannot be changed.

## 5.5 Interrupts

P\_INT\_REQ is very similar to P\_NCBWR\_REQ with the following differences.

The USC will check to see if the slave's interrupt queue has enough space for the interrupt. If yes, then it will transfer the interrupt packet from the master to the slave. If not, it issues an S\_INAK to the master, who will have to retry the interrupt request later.

Also, if the target port is in sleep mode, the USC will begin the wakeup sequence for the target, and issue S\_INAKs to the master until the SLEEP bit in the USC for that slave has been cleared.

## 5.6 Flow Control

The sizes of the USC's master request queues are listed in Table 5-2, "USC Master Queue Sizes."

Table 5-2 USC Master Queue Sizes

Queue Name	Queue Depth is 2-Cycle Packets	Note
MRQ0/CPU	1	Master request queue for CPU class 0
MRQ1/CPU	4	Master request queue for CPU class 1
MRQ/SYSIO	2	Master request queue for the U2S

All transactions issued by the U2S collapse into a single master request queue, regardless of the class bit in the transaction. The USC expects all requests from the U2S to be issued from class 0.

The USC performs flow control by knowing the length of the PRequest, data, and interrupt queues at the slave, per the UPA Architecture. These are programmed into the SC\_Port\_Config registers by the boot processor after power on. The USC will never issue more requests to a slave than it has room.

The maximum number of outstanding transactions varies substantially and is generally determined by the queue sizes in the USC and at the UPA ports. However, the following restrictions are always enforced in the USC, regardless of these queue sizes.

- One outstanding memory transaction
- One transaction per class that the Port Interface detects as an error



## 5.7 Blocking Conditions

The blocking conditions or blocking rules are used to maintain ordering and to disallow parallelism whenever a condition occurs which could cause things to get out of order. S\_Replys for transaction belonging to a master class are strongly ordered. The blocking rules are implemented in the PIF and are described below.

### 5.7.1 Definitions

A transaction is active (or pending) from the time the USC determines that the transaction does not need to be blocked and can proceed to carry out the transaction to the time the S\_Replys are sent for this transaction.

The SRequest outstanding flag is set when a transaction which needs to send an SRequest becomes active, and is reset when the P\_Reply for the SRequest is received.

The U2S RMW flag is set when the U2S P\_RDO\_REQ becomes active and is reset when the U2S P\_WRB\_REQ becomes inactive.

The cancel flag is set when the USC receives a P\_SACKD in response to a S\_CPI\_REQ or S\_INV\_REQ and is reset when the corresponding P\_WRB\_REQ is cancelled (when the S\_WBCAN is sent out).

### 5.7.2 Blocking for Noncached Transactions

A noncached transaction from any class is blocked if:

1. there is a preceding read transaction and the requested transaction is a write For reads, the USC cannot issue the S\_Replys until the slave has sent its P\_Reply to the USC. However, when the USC forwards a write to the slave, it also sends the S\_Reply to the master at the same time to begin sourcing data onto the bus. So to avoid getting S\_Replys out of order, the USC has to wait for the read to finish before issuing the write.
2. there is a preceding transaction going to a different slave from the same master class. Because each slave has an unpredictable amount of reply latency this has to be done to keep S\_Replys in order.
3. there is a preceding cached transaction from the same master class. The rationale is the same as that for the preceding paragraph 2.
4. the slave queue for the addressed slave is full.

### 5.7.3 Blocking Conditions for Cached Transactions

1. There cannot be more than one active transaction with the same index. Before each transaction becomes active, its index is checked against the indices of all active transactions in the coherence scoreboard. If there is a match, the transaction is blocked.
2. The processor cannot access the same block that the U2S is doing an RMW on. This is part of the no-data tags rules. When the U2S RMW flag is set, a processor transaction with the same index is blocked.
3. There can only be one outstanding SRequest at any one time. This is an UPA requirement. No SRequest can be sent if the SRequest outstanding flag is set. The USC handles the case of two transactions wanting to send SRequests in the same clock cycle by giving the U2S priority over a processor P\_WRI\_REQ with IVA bit set.
4. A cached transaction is blocked if there are outstanding non-cached transactions from the same class. This is done to ensure that the S\_Replys are kept in order.
5. There can only be one transaction accessing memory at any one time. The memory controller can only handle one transaction at a time, and can become blocked under the following conditions:
  - A. the memory controller (MC) is busy performing a refresh operation
  - B. the MC is busy processing an earlier read or write transaction
  - C. the MC is waiting for the read buffer in the XB1 for a previous read to be drained by the DPS. This can be caused either by scheduling delays or by the need to wait for an P\_Reply for an SRequest sent to the CPU

## 5. Packet Handling

For cached transactions, the two main resources required are memory and the SRequest slot. Table 5-3 shows how each transaction uses those resources.

*Table 5-3* Resource Usage for Cached Transactions

Transaction	Memory	SRequest Slot
CPU read	Yes	No
CPU WRB	Yes	No
CPU WRB (cancelled)	No	No
CPU WRI (IVA = 0)	Yes	No
CPU WRI (IVA = 1)	Yes	Yes
U2S RDO/RDD <sup>[1]</sup>	Yes	Yes
U2S WRB	Yes	No
U2S WRI	Yes	Yes

1. In this case, both resources are used simultaneously.

## 5.8 Class Symmetry

### 5.8.1 Processor

The USC does not implement perfect class symmetry; that is, it assumes that certain transactions are always issued in certain classes from the processor. In particular, the USC will only support the transaction to class allocation that UltraSPARC-1 processor actually implements.

1. **CLASS0:** P\_RDS\_REQ, P\_RDSA\_REQ, P\_RDO\_REQ, P\_RDD\_REQ, and P\_NCBRD\_REQ.
2. **CLASS1:** P\_WRI\_REQ, P\_NCBWR\_REQ, P\_INT\_REQ, P\_NCWR\_REQ, P\_NCRD\_REQ, and P\_WRB\_REQ.

This is the only allocation that the USC is guaranteed to support.

### 5.8.2 U2S

The USC only implements a single master class for the U2S, and this is class 0. Requests issued from class 1 in the U2S will have undefined results.

## 5.9 Presence Detection

The USC assumes that both the processor and the U2S are always present. It does not perform any type of presence detection for these two UPA clients, since a system without either one would not function, and both are hard-wired into the UltraSPARC-I reference platform system board.

For the UPA64S client, the USC will examine UPA\_PREPLY2[1:0] as soon as it comes out of reset. If it sees that these lines are a non-zero value, then it concludes that there is no UPA64S client present. Any read transaction sent to this client, cached or noncached, will result in an S\_ERR being returned to the master, and the IADDR bit being set. An P\_WRB\_REQ issued to UPA64S will have undefined results regardless of whether UPA64S client is present or not. Therefore it is important that the UPA64S client drive its P\_Reply lines to an idle value during reset and after coming out of reset.

### 5.10 DATA\_STALL

The USC has two data stall signals, UPA\_DATA\_STALL0 (for the processor) and UPA\_DATA\_STALL1 (for the U2S), which are used to regulate the flow of data when accessing a slower device (such as memory) or a device on a narrower bus.

DATA\_STALL is only asserted for block transfers; it is never asserted for single-cycle transfers.

DATA\_STALL is only sent to the recipient of the data transfer. The UPA port that is sourcing the data will never see its DATA\_STALL asserted.

Table 5-4 shows the conditions under which the DATA\_STALL is asserted.

Table 5-4 DATA\_STALL Assertion

Master	Operation	Slave	DATA_STALL Asserted	Note
CPU	Read	Memory	UPA_DATA_STALL0	After 32 bytes of data have been delivered
CPU	Write	Memory	No	
U2S	Read	Memory	UPA_DATA_STALL1	After 32 bytes of data have been delivered
U2S	Write	Memory	No	
CPU	Read	U2S	UPA_DATA_STALL0	
CPU	Write	U2S	No	
U2S	Read	CPU	No	
U2S	Write	CPU	UPA_DATA_STALL0	
CPU	Read	FFB	UPA_DATA_STALL0	
CPU	Write	FFB	No	
U2S	Read	FFB	No	
U2S	Write	FFB	No	

## ***5.11 Internal Arbitration Priorities***

### ***5.11.1 PIF***

There are several points of contention within the PIF block.

#### ***5.11.1.1 UPA Address Bus 0***

A round robin arbitration scheme is used as required by the UPA specification.

#### ***5.11.1.2 Coherence Scoreboard***

All coherent requests that go to memory must be entered into the scoreboard. Only one request is allowed to be added to the scoreboard at any one time. In case of multiple simultaneous requests the following priority is used:

1. U2S
2. CPU class 0
3. CPU class 1

#### ***5.11.1.3 Slave Request Queues***

Each slave request queue has an arbiter in front of it to accept requests. The priority used in case of multiple simultaneous requests is:

1. CPU class 1
2. U2S
3. CPU class 0

#### 5.11.1.4 UPA Output

There are four different internal sources for packets that need to be sent out to UPA address bus 0. A fixed priority scheme is used:

1. SRequest due to the U2S request
2. SRequest due to CPU request
3. Forwarded PRequest to the U2S
4. Forwarded PRequest to CPU

#### 5.11.2 Data Path Scheduler

The data path scheduler (DPS) contains an arbiter which is used to determine which transaction to schedule. The DPS uses a modified round robin scheme. There are four sources of requests inside the DPS: MemUnit (for memory requests), FfbUnit (for requests to FFB), UpaUnit (for processor <-> U2S transfers) and ErrUnit (for transactions resulting in a error). The DPS maintains a round robin pointer that is incremented every time a request is scheduled. If the pointer is pointing to a valid request, then the arbiter will select that source. If not, then the following fixed priority is used:

1. MemUnit
2. FfbUnit
3. UpaUnit
4. ErrUnit

The DPS always inserts a dead cycle between data packets. The only exception to this is back-to-back write singles from CPU to FFB. The DPS will schedule these transfers so that there is no dead cycle between the data packets.

#### 5.11.3 Memory Controller

If the memory controller (MC) receives both a refresh request and a read/write transaction from PIF in the same clock cycle, the refresh request has first priority.

### 5.12 Concurrency

The DPS allows a limited amount of concurrency. If the 64-bit data path is busy, then it will allow a memory transaction from the CPU, which is on the 128-bit data path, to be scheduled.

Likewise, if the 128-bit data path is busy, then it will allow a memory transaction from U2S, which is on the 64-bit data path, to be scheduled.

### 5.13 Reserved P\_Replies

If the USC receives a P\_Reply from a slave that is documented in the “UPA Interconnect Architecture” as being “reserved,” then the results will be undefined.

### 5.14 Error Handling

#### 5.14.1 Fatal Errors

These are the only errors that are detected by the USC.

1. *Parity error on UPA address bus 0.* This is logged in the IPRTY bit in the SC\_Port\_Status register of the corresponding master. A POR (power-on reset) is generated.
2. *The USC receives a P\_FERR from the CPU or the U2S.* A POR is generated. P\_FERR can be issued at any time by a slave.
3. *Master queue overflow.* This is a fatal error, and is logged in the MCxOF bit in the appropriate SC\_Port\_Status register. A POR (power-on reset) is generated.

#### 5.14.2 Nonfatal Errors

Nonfatal errors are those errors which are logged in the IADDR or IPORT bit in the SC\_Port\_Status.

1. *Access to a unimplemented address.* This is logged in the IADDR bit in the SC\_Port\_Status register of the corresponding master. This is not a fatal error. This includes accesses to the UPA64S client when no client is detected to be present.
2. *Access to a port that is not present by a misdirected interrupt request.* This is logged in the IPORT bit in the SC\_Port\_Status register of the corresponding master. This is not a fatal error. In the USC, only the processor is allowed to receive interrupt packets.
3. *Coherent read requests to the FFB.* This is logged in the IADDR bit.
4. *Noncached accesses to memory.* This is logged in the IADDR bit.

Some of these errors are the result of programming errors. The USC implements the error-handling philosophy as described in the UPA Architecture document. These errors are not allowed to hang the system and are always completed end to end. The data for any write to a nonexistent port is dropped on the floor, but the transfer completes normally.



## 5.15 Timing Diagrams

### 5.15.1 Noncached Transactions

All data transfers have a dead cycle between them, except for back-to-back single writes from the processor to the FFB. Figure 5-1, "Best-Case PRequest-to-PRequest Timing (ABus0 to ABus0)." through Figure 5-15, "Best-Case PRequest-to-Memory Request Timing (Normal Path)." show some of the best-case response timing for the USC.

The only UPA masters in the system reside on address bus 0 (processor and U2S), so PRequests can only originate there. But depending on the destination, they can be forwarded to either address bus 0 or 1.

Figure 5-1, "Best-Case PRequest-to-PRequest Timing (ABus0 to ABus0)." shows the best-case timing for forwarding an PRequest on Address Bus 0.

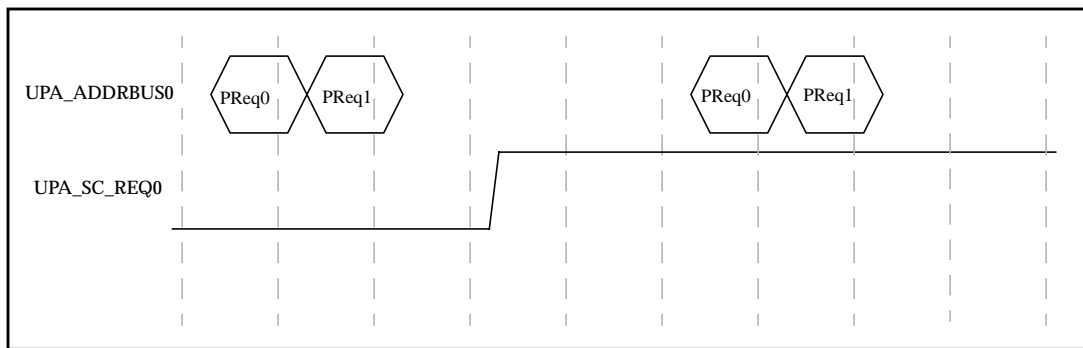


Figure 5-1 Best-Case PRequest-to-PRequest Timing (ABus0 to ABus0)

Figure 5-2, "Best-Case PRequest-to-SRequest Timing (ABus0 to ABus0)." shows the best-case timing for sending an SRequest that is triggered by an PRequest. This timing diagrams applies to both SRequests triggered by P\_WRI\_REQ with the IVA bit set issued from the processor as well as coherent requests from the U2S.

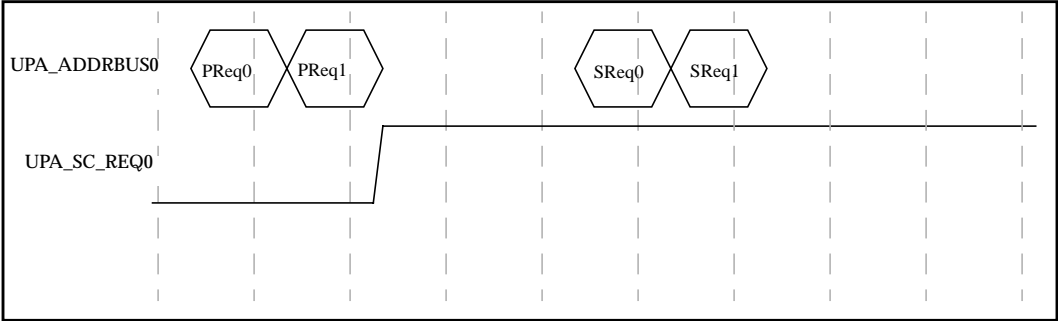


Figure 5-2 Best-Case PRequest-to-SRequest Timing (ABus0 to ABus0)

Since the USC is always the master of Address Bus 1 and no arbitration is ever required on that bus, the time it takes for the USC to forward a packet from Address Bus 0 to Address Bus 1 is faster, as shown in Figure 5-3, "Best-Case PRequest-to-SRequest Timing (ABus0 to ABus1)."

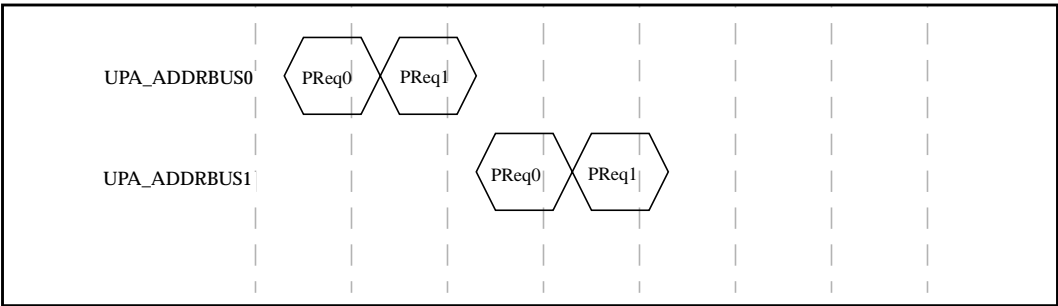


Figure 5-3 Best-Case PRequest-to-SRequest Timing (ABus0 to ABus1)

## 5. Packet Handling

Figure 5-4, "Best-Case Timing for Noncached Single Read, UPA64 -> UPA128." and Figure 5-5, "Best-Case Timing for Noncached Block Read, UPA64 -> UPA128." show the best-case timing for CPU NC single and block read from the U2S referenced to the time the P\_Reply is issued from the slave. The timing for CPU NC read from FFB is the same, except that it is referenced to a single-cycle P\_RASB instead of the two-cycle P\_RAS/P\_RAB.

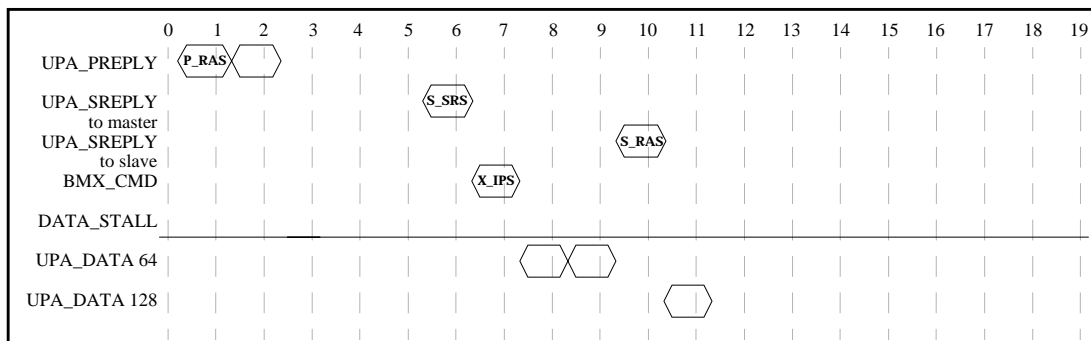


Figure 5-4 Best-Case Timing for Noncached Single Read, UPA64 -> UPA128

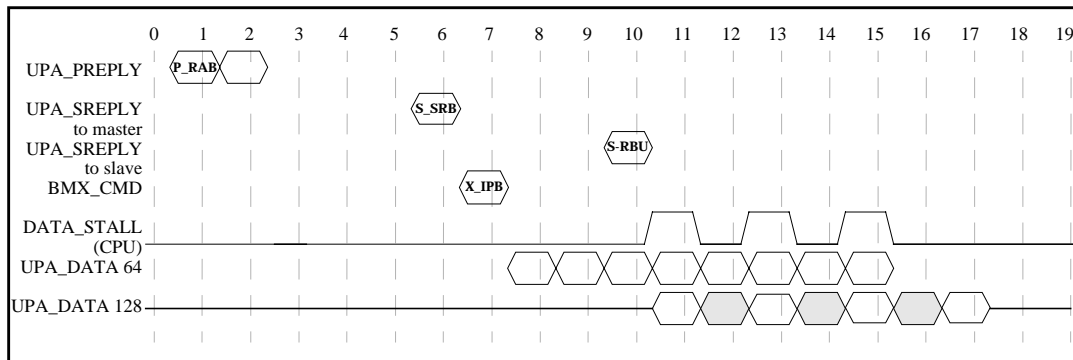


Figure 5-5 Best-Case Timing for Noncached Block Read, UPA64 -> UPA128

Figure 5-6, "Best-Case Timing for Noncached Single Write, UPA128 -> UPA64." and Figure 5-7, "Best-Case Timing for Noncached Block Write, UPA128 -> UPA64." illustrate the best-case timing for a CPU NC single and block write to the U2S. A CPU NC write to the FFB has similar timing, except that the time it takes to forward the PRequest is slightly less.

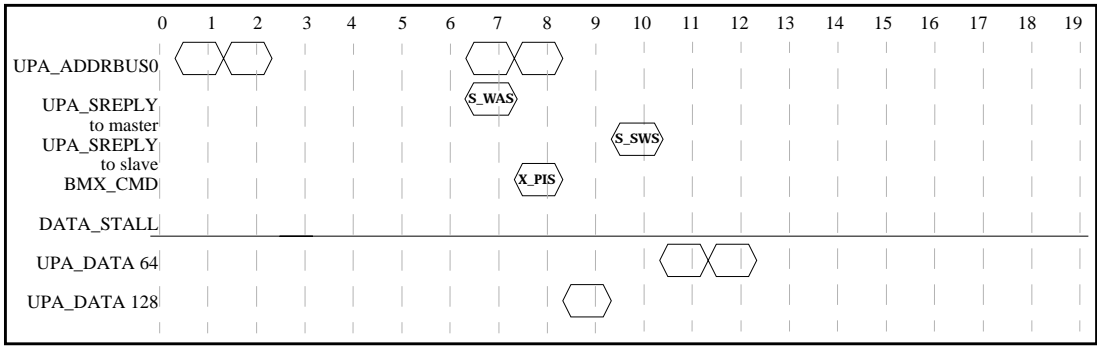


Figure 5-6 Best-Case Timing for Noncached Single Write, UPA128 -> UPA64

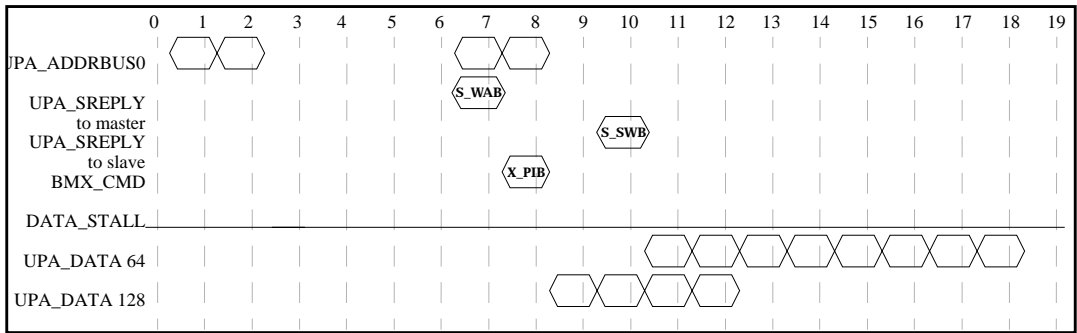


Figure 5-7 Best-Case Timing for Noncached Block Write, UPA128 -> UPA64

## 5. Packet Handling

Figure 5-8, "Best-Case Timing for Noncached Single Read, UPA64 -> UPA64." and Figure 5-9, "Best-Case Timing for Noncached Block Read, UPA64 -> UPA64." illustrate a U2S single and block read from the FFB, referenced to the P\_Reply from the FFB.

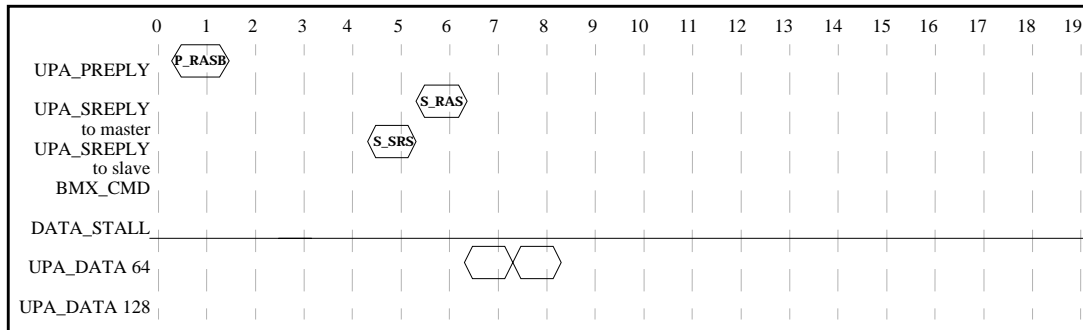


Figure 5-8 Best-Case Timing for Noncached Single Read, UPA64 -> UPA64

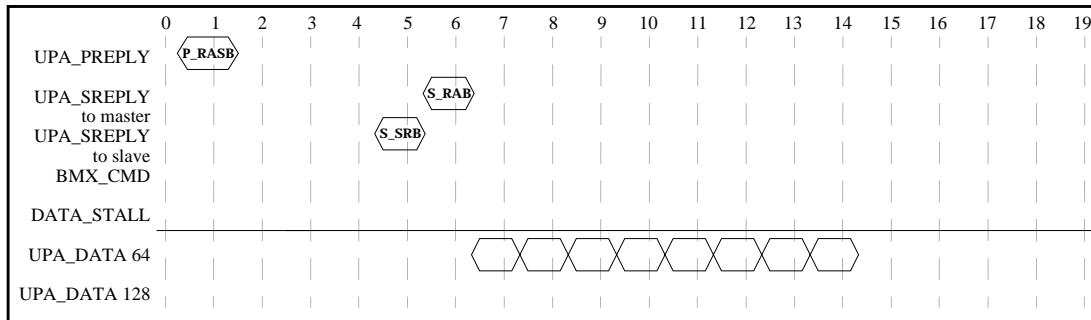


Figure 5-9 Best-Case Timing for Noncached Block Read, UPA64 -> UPA64

Figure 5-10, "Best-Case Timing for Noncached Single Write, UPA64 -> UPA64." and Figure 5-11, "Best-Case Timing for Noncached Block Write, UPA64 -> UPA64." illustrate the best-case timing for a U2S NC single and block write to FFB.

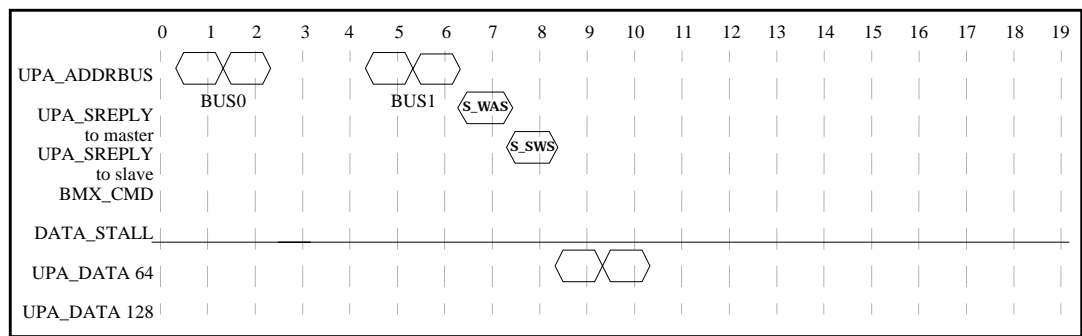


Figure 5-10 Best-Case Timing for Noncached Single Write, UPA64 -> UPA64

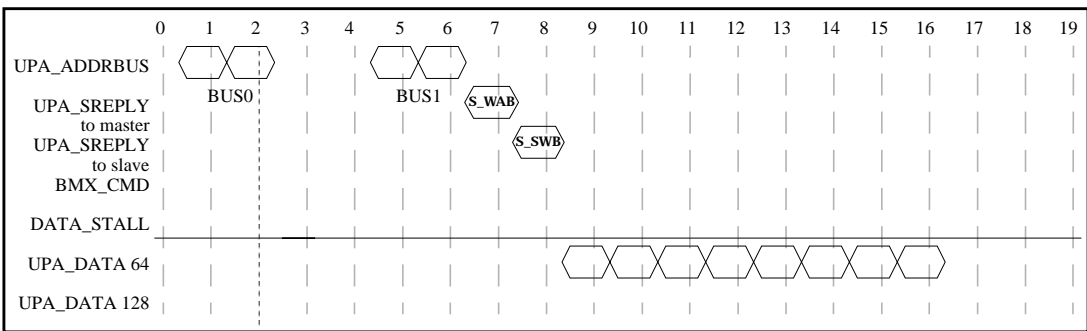


Figure 5-11 Best-Case Timing for Noncached Block Write, UPA64 -> UPA64

## 5. Packet Handling

Figure 5-12, "Best-Case Timing for Noncached Single Read, UPA128 -> UPA64." and Figure 5-13, "Best-Case Timing for Noncached Block Read, UPA128 -> UPA64." illustrate the nominal timing for a U2S NC single and block read from the processor.

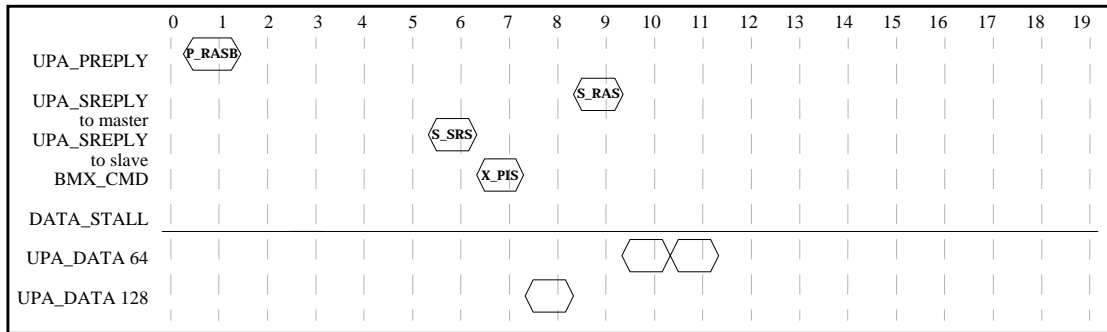


Figure 5-12 Best-Case Timing for Noncached Single Read, UPA128 -> UPA64

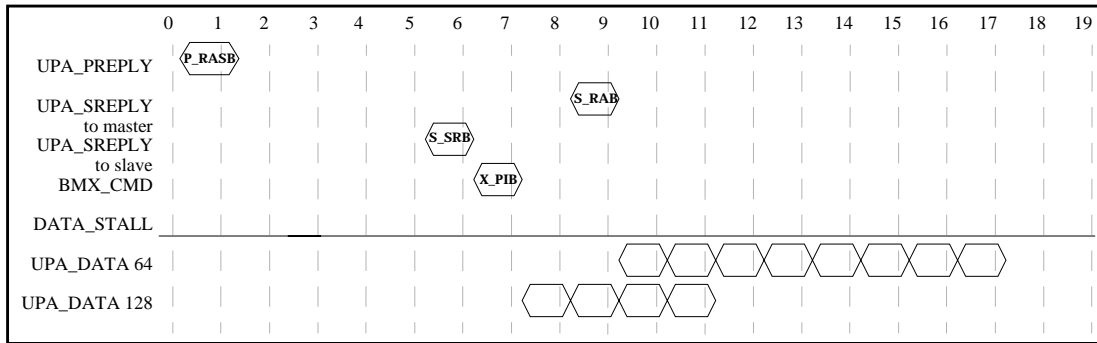


Figure 5-13 Best-Case Timing for Noncached Block Read, UPA128 -> UPA64

Timing diagrams for U2S NC single and block write to the processor are not shown because such transactions are dropped by the USC.

## 5.15.2 Coherent Transactions

Timing diagrams for coherent memory transactions are in Chapter 6, "Memory System."

### 5.15.3 Fast Path

Figure 5-14, "Best-Case PRequest-to-Memory Request Timing, Read (Fast Path)." shows the "fast path" timing for memory reads issued from the processor. The fast path is only available for reads issued from the processor's master class 0; it is not available for writes or for any accesses from the U2S. Fast path can only be used if there are no coherent transactions outstanding. Fast path is not implemented for processor writes because for P\_WRI\_REQ we need to examine the IVA bit before launching the request to memory, and the IVA bit is in the second half of the PRequest packet. P\_WRB\_REQ almost always follows a victimizing read, and this can be overlapped with the read. Accesses from the U2S are less latency sensitive. Since the fast path is very timing critical and adds additional complexity to the logic, it is not implemented for the U2S.

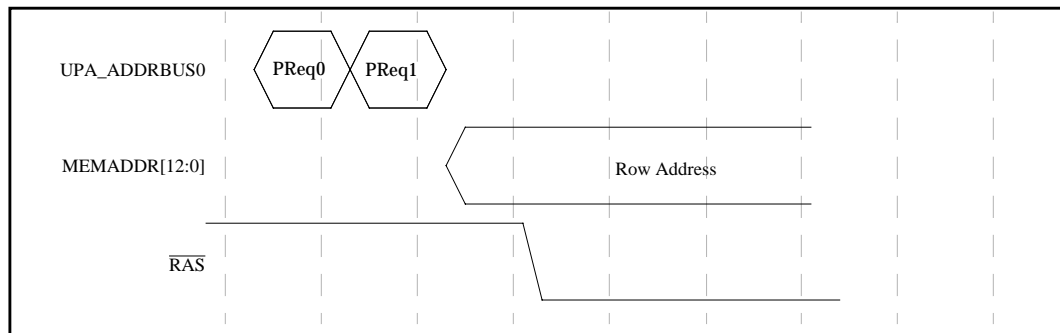


Figure 5-14 Best-Case PRequest-to-Memory Request Timing, Read (Fast Path)



Figure 5-15, "Best-Case PRequest-to-Memory Request Timing (Normal Path)." shows a read or write issued through the "normal path."

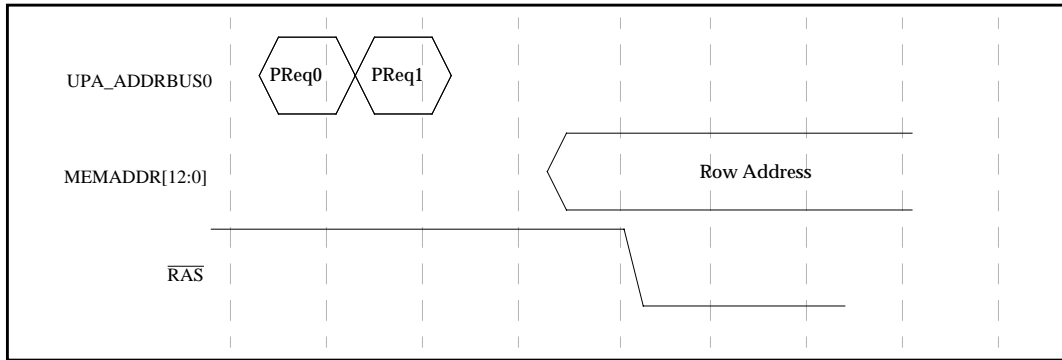


Figure 5-15 Best-Case PRequest-to-Memory Request Timing (Normal Path)



## 6.1 Introduction

This section describes the structure and operation of the reference platform memory system.

## 6.2 SIMMs

The SIMMs that are supported in the reference platform memory system are shown in Table 6-1, "SIMMs Supported by the USC."

Table 6-1 SIMMs Supported by the USC

SIMM Size	Base Device	Number of Devices	Note
16 MByte	4 megabit (Mb) (1Mb x 4)	36	
32 MByte	16 Mb (2 Mb x 8)	18	The USC will only support parts with 2 K rows and 1 K columns.
64 MByte	16 Mb (4 Mb x 4)	36	The USC will only support 4 K refresh parts (4 K rows, 1 K columns).
128 MByte	64 Mb (8 Mb x 8)	18	The USC only supports parts with 8 K rows and 1 K columns.

The reference platform uses the faster version of the SPARCstation 20, DSIMM, which uses 60-ns DRAMs. The memory controller will not work with the older version of DSIMM which use 80-ns DRAMs.

The USC's memory controller (MC) can handle up to eight SIMMs. The minimum memory configuration is 32 MBytes, or two 16 MByte SIMMs. The maximum memory configuration is one gigabyte or eight 128 MByte SIMMs. This is explained further in the following sections.

### 6.3 Block Diagram

Figure 6-1, "Memory System Interconnection," shows the interconnection between the USC and the memory system.

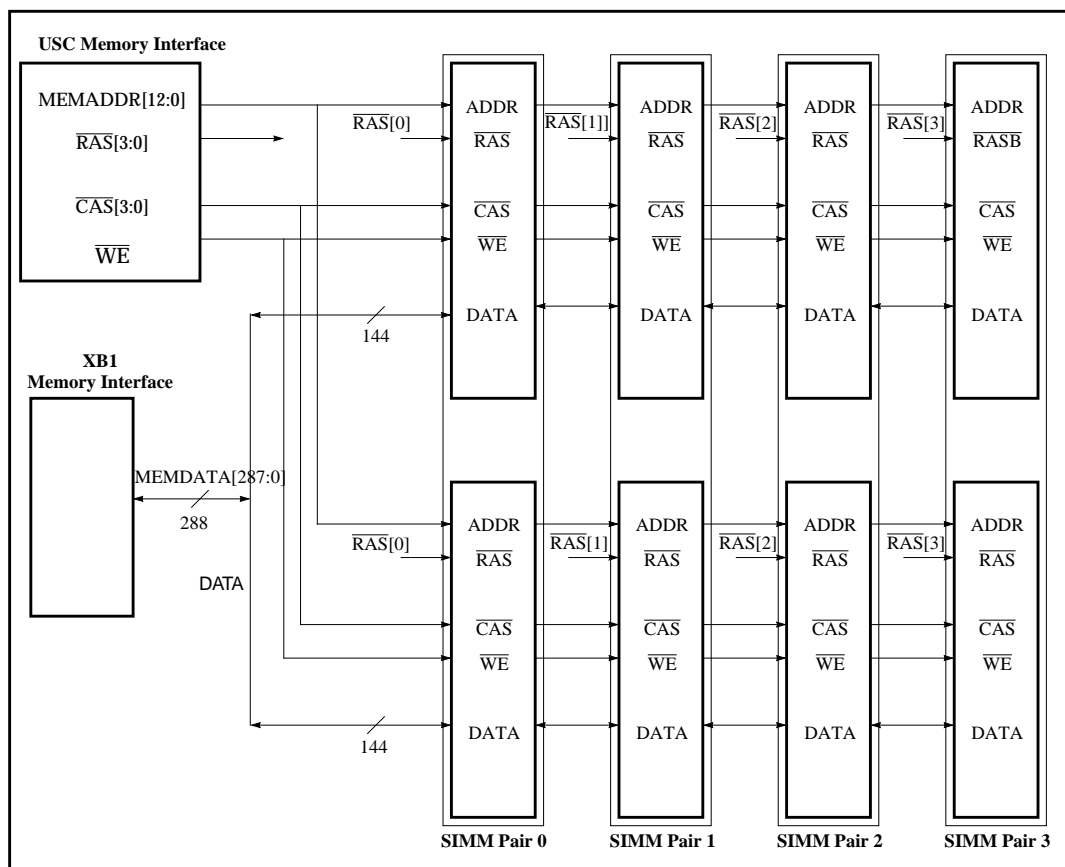


Figure 6-1 Memory System Interconnection

The memory data bus is 288 bits wide: 256 bits of data and 32 bits of ECC. Since each SIMM supplies only 144 bits of data, two SIMMs are required to fill the data bus. Therefore the minimum memory increment is 32 MBytes (2 x 16 MBytes).

## 6. Memory System

SIMMs are always accessed in pairs. When populating memory, it is important to ensure that SIMM pairs are always filled, and that both SIMMs in a pair are of the same type. Failure to follow these rules may result in memory waste or an inoperative system.

Because there are a total of eight SIMM slots, there can be a total of up to four SIMM pairs.

MEMADDR[12:0] and  $\overline{WE}$  are broadcast to all eight SIMMs.  $\overline{CAS}[3:0]$  are four replicated copies of the same signal. However, since each SIMM contains two  $\overline{CAS}$  inputs (one for each 72-bit half), there are also a total of eight loads on each  $\overline{CAS}$ .

The  $\overline{RAS}$  signals are radial. Each SIMM pair receives a copy of  $\overline{RAS}$ . Since each SIMM has two  $\overline{RAS}$  pins for each stack, there are effectively four loads on each  $\overline{RAS}$  line.

All address,  $\overline{RAS}$ ,  $\overline{CAS}$ , and  $\overline{WE}$  signals are buffered by the SIMM before being distributed to the DRAM array.

## 6.4 Addressing

Figure 6-2, "Addressing," shows the addressing scheme that is used by the memory controller.

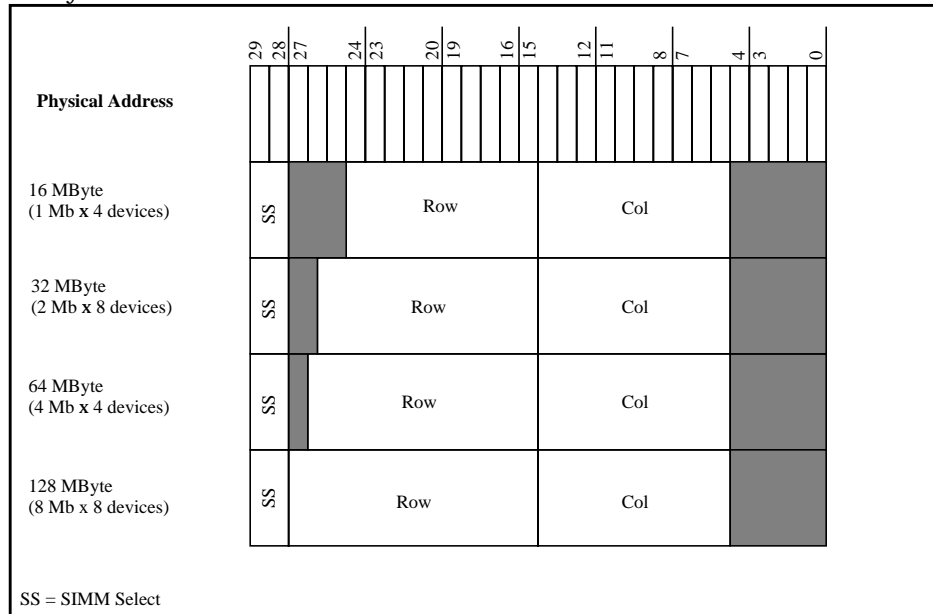


Figure 6-2 Addressing

In this scheme, PA[29:28] is used as a SIMM select, it selects which SIMM pair is to be accessed. The way that PA[29:28] maps into  $\overline{\text{RAS}}$  is shown in Table 6-2, "PA[29:28] to RAS Mapping."

Table 6-2 PA[29:28] to  $\overline{\text{RAS}}$  Mapping

PA[29:28]	$\overline{\text{RAS}}$ Asserted
00	$\overline{\text{RAS}}[0]$
01	$\overline{\text{RAS}}[1]$
10	$\overline{\text{RAS}}[2]$
11	$\overline{\text{RAS}}[3]$

## 6. Memory System

PA[27:15] always map into the row address (MEMADDR[12:0]) and PA[14:5] always map into the column address (MEMADDR[9:0]). This makes the internal muxing logic and the probing algorithm very simple. However, this also dictates that any DRAM used to build a SIMM must have a 10-bit column address.

The MC will only support 16 MByte devices that have a 4K organization (4K rows, 1K columns). This scheme does not support 2K parts (2K rows, 2K columns).

The MC will only support 64 MByte devices that have an 8K organization (8K rows, 1K columns).

PA[5:4] are used to determine wrapping (which 144-bit quantity to deliver first).

Since the memory block size in the reference platform is 64 bytes, and the memory data bus is only 256 bits or 32 bytes wide, a page-mode cycle is done to get the second half of the memory block. The least significant bit of the column address will toggle during the page-mode cycle. PA[5] will appear as the least significant bit of the column address, and PA[4] is used to tell the XB1 which half of the 32 bytes to deliver first.

Memory accesses are tagged with the master ID and class bit of the initiator. The two-bit master ID is carried on MEMADDR[12:11] and the class bit is carried on MEMADDR[10] during the column-address cycles. The only legal tags are:

- 000: access from CPU master class 0
- 001: access from CPU master class 1
- 110: access from the U2S master class 0

Any other tag should be considered an error.

Table 6-3, "Memory Address Map," illustrates the same information as Table 6-2, "PA[29:28] to RAS Mapping," but in a different form.

Table 6-3 Memory Address Map

SIMM Number	SIMM Type	Address Range (PA[29:0])
0	16 MByte	0x0000_0000 to 0x01ff_ffff
0	32 MByte	0x0000_0000 to 0x03ff_ffff
0	64 MByte	0x0000_0000 to 0x07ff_ffff
0	128 MByte	0x0000_0000 to 0x0fff_ffff
1	16 MByte	0x1000_0000 to 0x11ff_ffff
1	32 MByte	0x1000_0000 to 0x13ff_ffff
1	64 MByte	0x1000_0000 to 0x17ff_ffff
1	128 MByte	0x1000_0000 to 0x1fff_ffff
2	16 MByte	0x2000_0000 to 0x21ff_ffff
2	32 MByte	0x2000_0000 to 0x23ff_ffff
2	64 MByte	0x2000_0000 to 0x27ff_ffff
2	128 MByte	0x2000_0000 to 0x2fff_ffff
3	16 MByte	0x3000_0000 to 0x31ff_ffff
3	32 MByte	0x3000_0000 to 0x33ff_ffff
3	64 MByte	0x3000_0000 to 0x37ff_ffff
3	128 MByte	0x3000_0000 to 0x3fff_ffff

The USC will not detect accesses which go to an out-of-range address or to a non-existent SIMM. An access to memory will either wrap back to a SIMM which does exist, or return invalid data.



## 6.5 Refresh

$\overline{\text{CAS}}$ -before- $\overline{\text{RAS}}$  refresh is used.

The USC uses a distributed refresh strategy where the MC cycles through the SIMMs and issues refresh operations periodically. Only a single pair of SIMMs is refreshed at a time, so in a full memory system, there are four SIMM pairs to cycle through.

The MC uses the contents of the SIMM present vector in MC\_Control0 to determine which SIMMs to refresh. The MC will not refresh a SIMM pair unless its corresponding bit in the SIMMPresent vector is set.

The MC uses the contents of the RefInterval field in MC\_Control0 to determine the interval between refreshes. Each least-significant bit (LSB) of RefInterval corresponds to eight system clocks. The timer will periodically signal the need for a refresh. When the timer issues a refresh request, it will automatically reset itself and begin counting again, even before the refresh request has been honored. This ensures refreshes do not accumulate any additional delay due to the fact the refreshes cannot always be serviced immediately.

The MC uses a very simple algorithm to arbitrate between refreshes and memory accesses. If a memory read or write is going on, then the MC will wait for the read or write to finish before issuing the refresh request. The MC will never interrupt a read or write to issue a refresh. If a read or write reaches the MC the same clock cycle as a refresh request, then the refresh request has highest priority.

The MC uses the contents of MC\_Control1, in particular the CSR,  $\overline{\text{RAS}}$ , and RP fields, to determine the actual timing for the refresh operation.

After power-on, refresh is disabled. Setting the RefEnable bit in MC\_Control0 will allow refreshes to begin. RefInterval and SIMMPresent should be set to the proper values before setting the RefEnable bit.

Refresh is disabled when the machine is powered up and has to be enabled by setting the RefEnable bit. After that, refresh will continue even after a software reset or a reset caused by a fatal error.

## 6.6 Timing Diagrams

### 6.6.1 Basic Memory Timing

Because the USC is expected to operate under a wide range of clock frequencies, the MC timing is programmable so that memory timing can be optimized for any given frequency. Memory refresh and access timing has been divided into a number of segments. The MC\_Control1 register contains eight fields which allow custom tailoring of any particular segment: CSR, WPC1, RCD, PC0, CP, PC1, RP, and  $\overline{\text{RAS}}$ . For a detailed description of these fields see Chapter 4, "Programming Model."

The timing diagram in Figure 6-3, "Basic Read/Write Timing," shows a generic template for a read or write transaction, how it is broken down into segments, and the corresponding fields in the MC\_Control1 which controls that particular segment. Each individual segment is programmable to certain values.

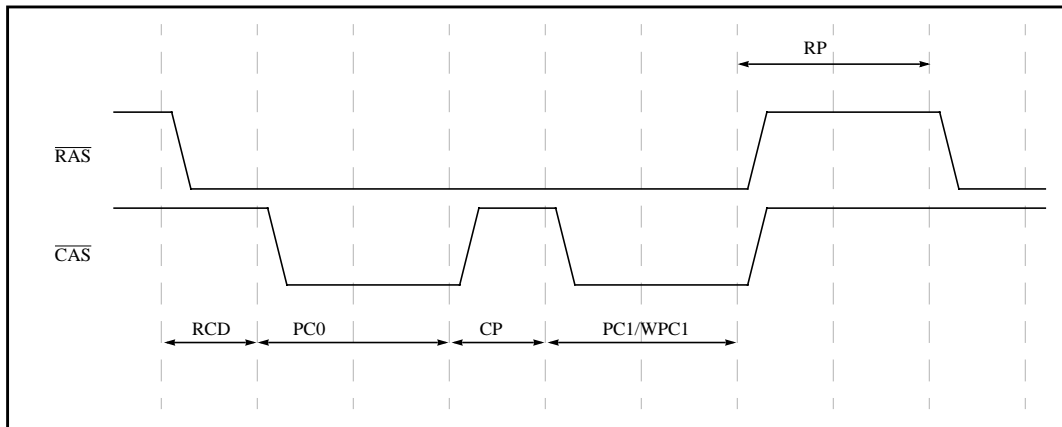


Figure 6-3 Basic Read/Write Timing

1. RCD is the  $\overline{\text{RAS}}$ -to- $\overline{\text{CAS}}$  delay
2. PC0 is the page cycle 0 time
3. CP is the  $\overline{\text{CAS}}$  precharge time
4. PC1 is the page cycle 1 time for a read operation
5. WPC1 is the page cycle 1 time for a write operation
6. RP is the  $\overline{\text{RAS}}$  precharge time

Reads use RCD, PC0, CP, PC1, and RP. Writes only use RCD, CP, WPC1, and RP. The PC0 is fixed.

The timing diagram in Figure 6-4, "Basic Refresh Timing," shows a generic template for a refresh operation.

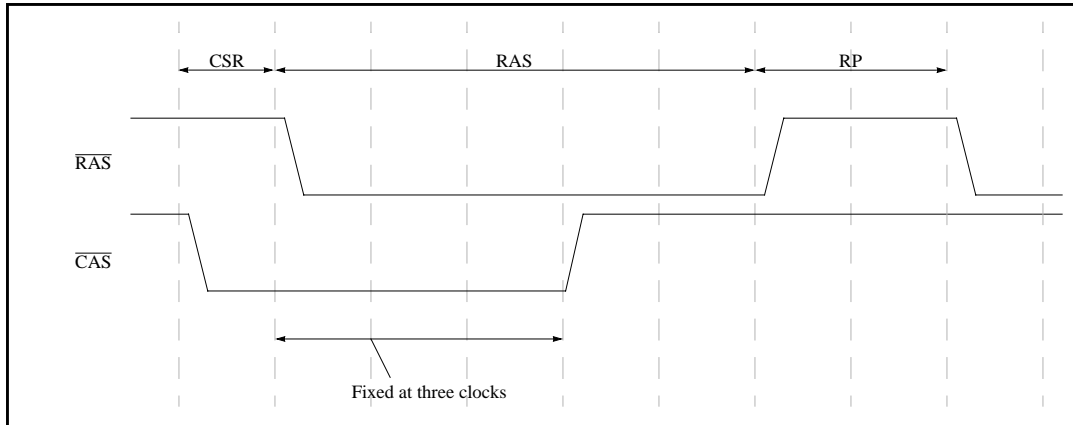


Figure 6-4 Basic Refresh Timing

1.  $\overline{\text{CSR}}$  is the  $\overline{\text{CAS}}$ -to- $\overline{\text{RAS}}$  assertion time
2.  $\overline{\text{RAS}}$  is the minimum  $\overline{\text{RAS}}$  timing
3. RP is the  $\overline{\text{RAS}}$  precharge timing

## 6.6.2 UPA-to-Memory Timing

Figure 6-5, "Best-Case UPA-to-Memory Timing (Fast Path)," and Figure 6-6, "Best-Case UPA-to-Memory Timing (Normal Path)," show the minimum time for a UPA memory request packet issued on the UPA address bus pins to traverse the USC and appear on the USC's memory outputs, assuming that the USC is idle.

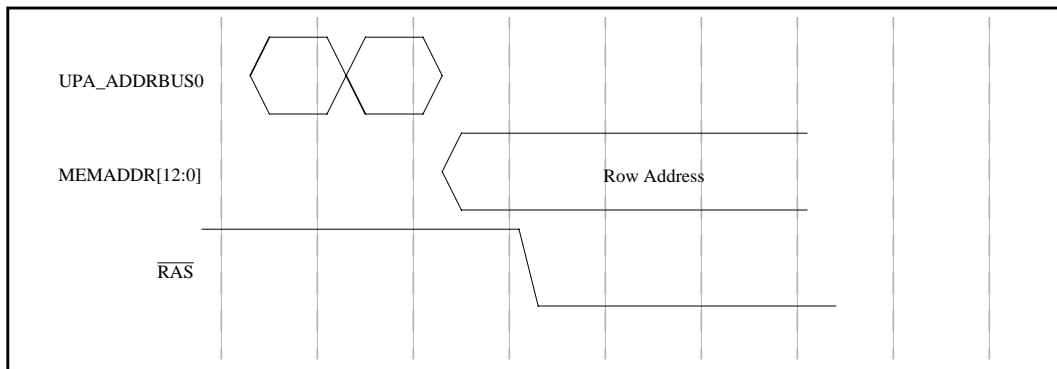


Figure 6-5 Best-Case UPA-to-Memory Timing (Fast Path)

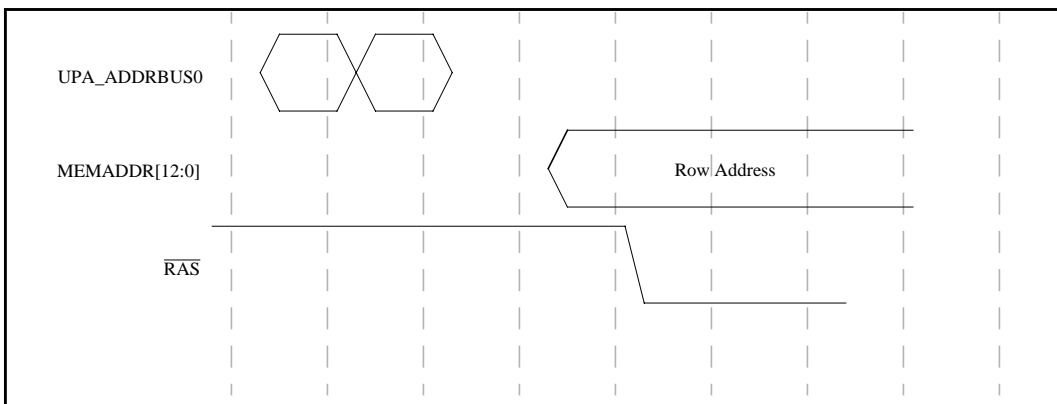


Figure 6-6 Best-Case UPA-to-Memory Timing (Normal Path)

Fast-path timing is only applicable for memory reads issued from the processor. All other memory accesses use the normal path. Fast path and normal path are explained in Chapter 5, "Packet Handling." Section 5.15.3, "Fast Path."

### 6.6.3 Memory Access Timings

This section illustrates read, write, and refresh timings for the following frequencies of interest.

- Default timing
- 83.3 MHz (12 ns)
- 71.4 MHz (14 ns)
- 66.7 MHz (15 ns)
- 62.5 MHz (16 ns)
- 55.5 MHz (18 ns)
- 50.0 MHz (20 ns)
- Minimum timing

Timings for accesses from both the CPU and the U2S are shown. The main difference between the two is that the CPU resides on a 128-bit-wide data bus and the U2S resides on a 64-bit-wide data bus, so the data transfer timing is different.

The timing diagrams show best-case idle timing. Care must be taken in interpreting these diagrams, since the timings shown here may not match exactly what might be seen during system operation. The reason is that the movement of data on the UPA buses and the movement of data on the memory bus (MWB) is somewhat decoupled by the read and write buffers in the XB1.

For memory reads, read data from the SIMMs can be loaded into the XB1 read buffer long before it is delivered to the master on the UPA data bus. Likewise, for memory writes, write data from the master can be loaded into the XB1 write buffer long before it is actually written into memory.

The signals can be divided into two groups. The first group consists of MEMADDR,  $\overline{\text{RAS}}$ ,  $\overline{\text{CAS}}$ ,  $\overline{\text{WE}}$ , MRB\_CTRL, and MWB\_CTRL, which are all generated by the Memory Controller. Their timing relationships are invariant and will match what is shown in these diagrams. The second group consists of UPA\_SREPLY, BMX\_CMD, and UPA\_DATA\_STALL, which are all generated by the Data Path Scheduler. Their timing relationships are invariant and will match what is shown in these diagrams. However, the relationships between the two groups are not invariant; they will vary depending on whether the data paths and the USC are idle or not, and whether it is a read or write transaction. The diagrams here only show what happens when everything is completely idle.

A precharge operation is performed between every memory access, regardless of whether the two accesses go to the same SIMM or to different SIMMs; there is no overlap of successive memory operations going to different SIMMs. The MC does not leave the SIMMs in page mode.

In general, back-to-back reads or back-to-back writes are separated only by the precharge time, assuming that the requests can be issued quickly enough at the address bus.

Table 6-4, "MC\_Control1 Timing Values." shows the register values to be used for the various operating frequencies.

---

**Note:** The values found in Table 6-4 are very important for the register programming of the memory controller. Included in the table are the programming data for the 100 MHz (10-ns) and the 90 MHz (11-ns).

---

Table 6-4 MC\_Control1 Timing Values

Clock Period (ns)	CSR	WPC1	RCD	PC0	CP	PC1	RP	$\overline{\text{RAS}}$
Default	1	11	1	11	1	11	11	11
10	0	01	1	11	0	11	10	10
11	0	01	1	11	0	11	01	10
12	0	00	1	10	0	10	01	10
14	0	00	1	01	0	01	00	01
15	0	00	0	10	0	01	00	01
16	0	00	0	10	0	01	00	00
18	0	00	0	01	0	01	00	00
20	0	00	0	01	0	00	00	00
Minimum	0	00	0	00	0	00	00	00

### 6.6.3.1 Default Memory Timing

Figure 6-7, "Default Memory CPU Read Timing," through Figure 6-11, "Default Refresh Timing," show the default timing after power-on. These are the slowest, most conservative timings possible and are guaranteed to work at any frequency.

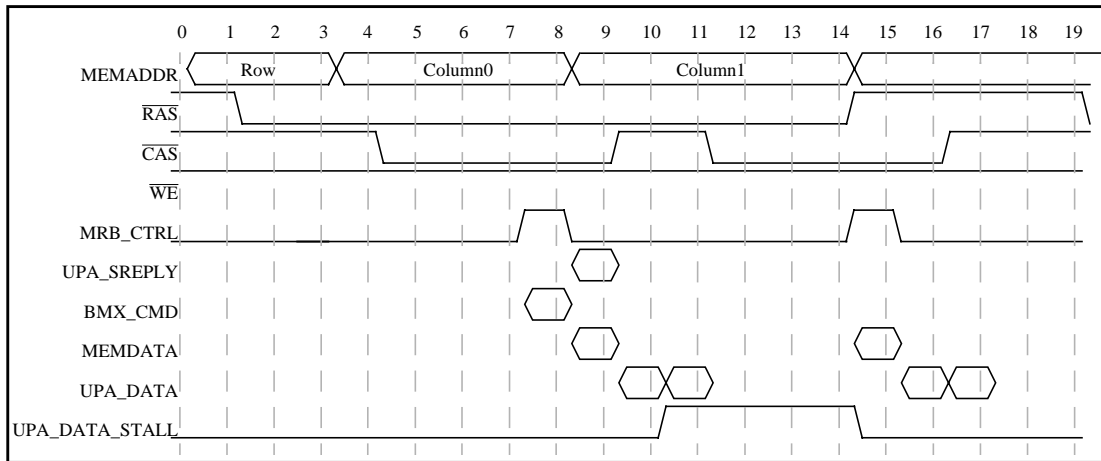


Figure 6-7 Default Memory CPU Read Timing

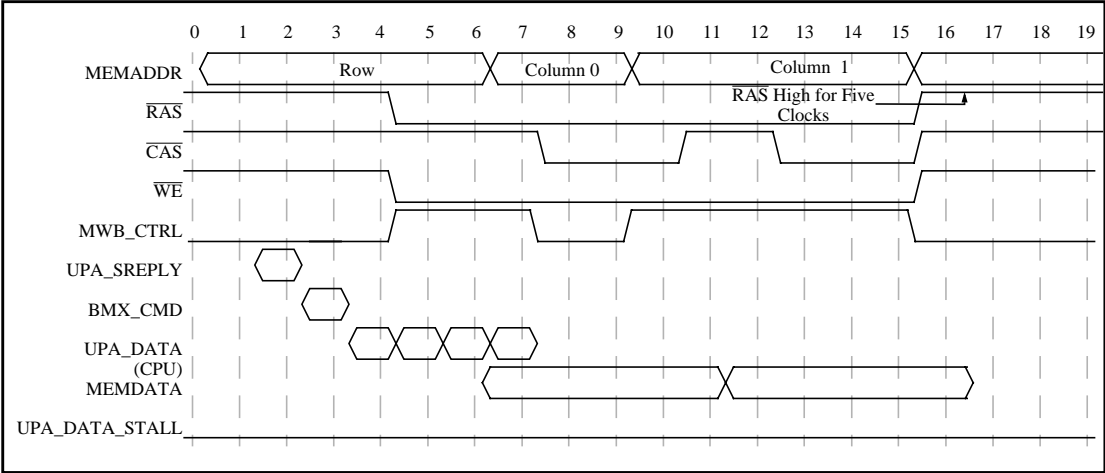


Figure 6-8 Default Memory CPU Write Timing

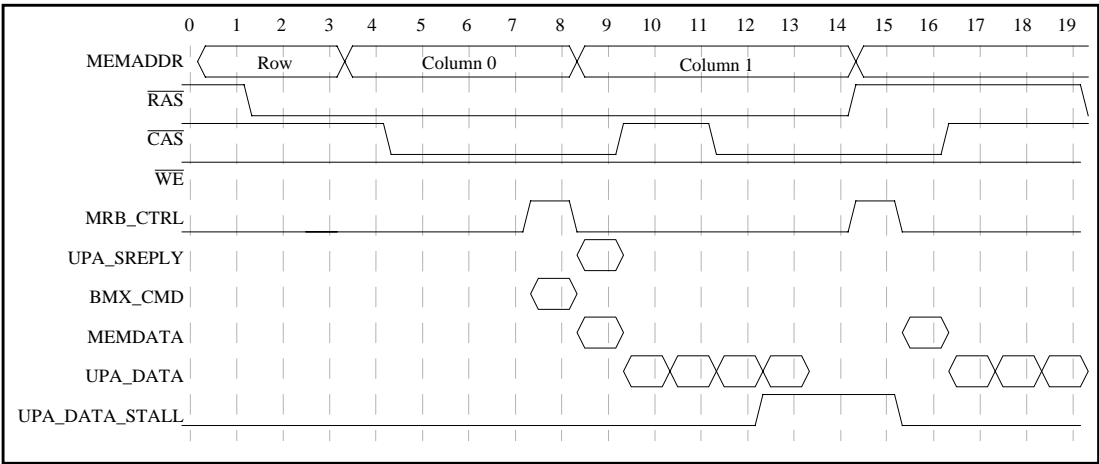


Figure 6-9 Default Memory U2S Read Timing



## 6. Memory System

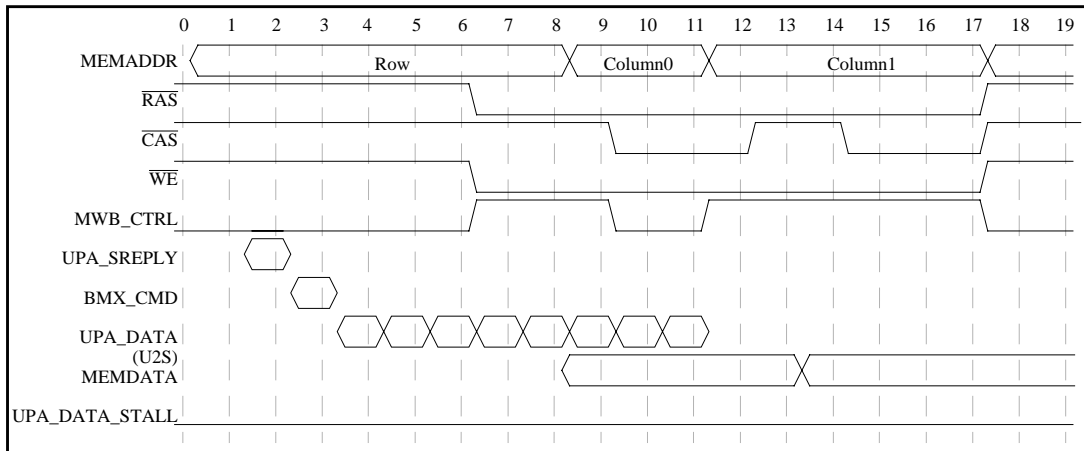


Figure 6-10 Default Memory U2S Write Timing

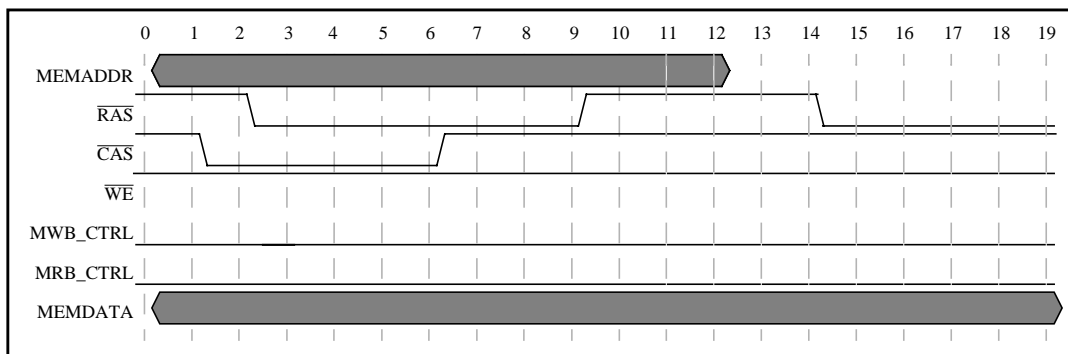


Figure 6-11 Default Refresh Timing

6.6.3.2 83.3-MHz (12-ns) Timings

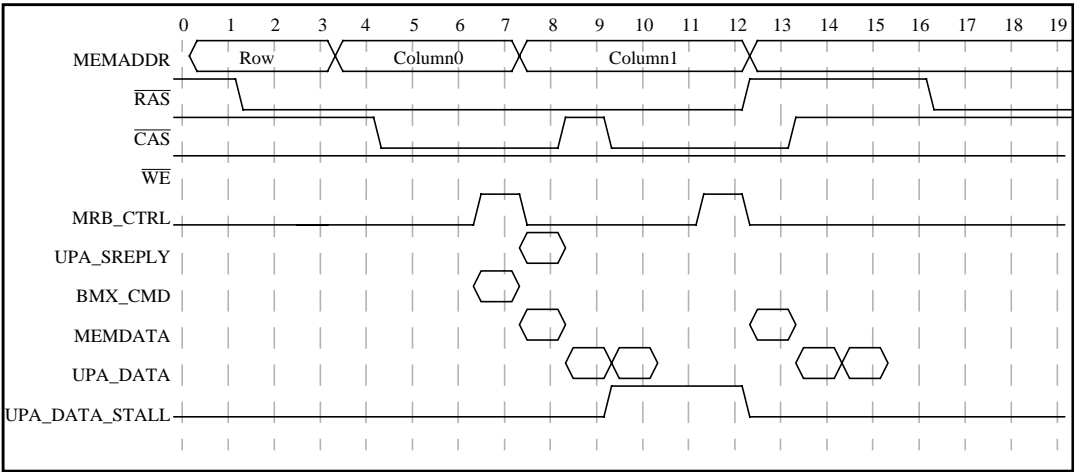


Figure 6-12 83.3-MHz CPU Read Timing

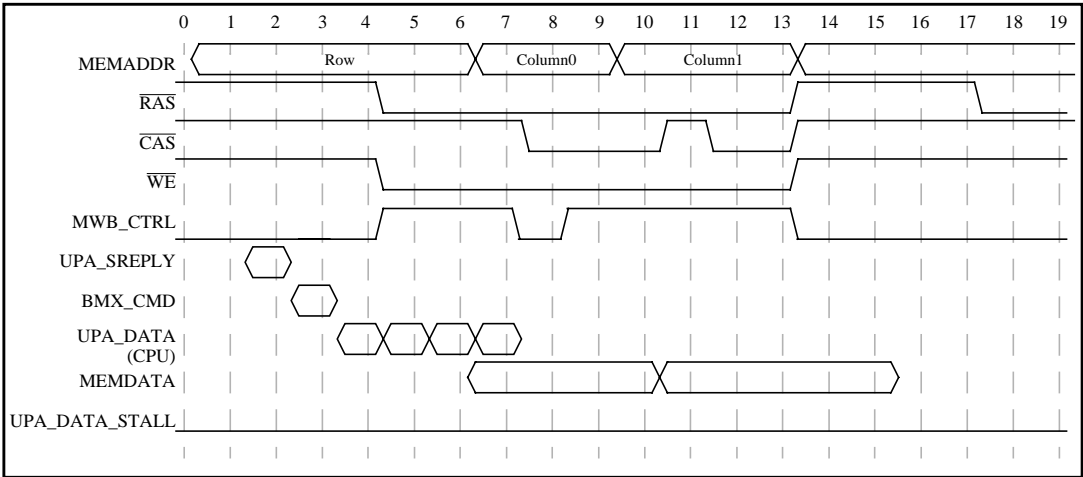


Figure 6-13 83.3-MHz CPU Write Timing

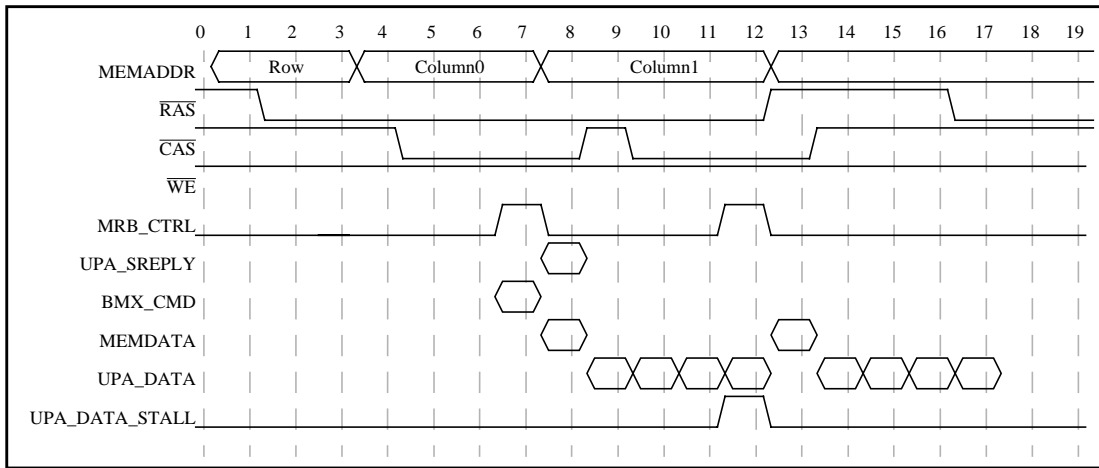


Figure 6-14 83.3-MHz U2S Read Timing

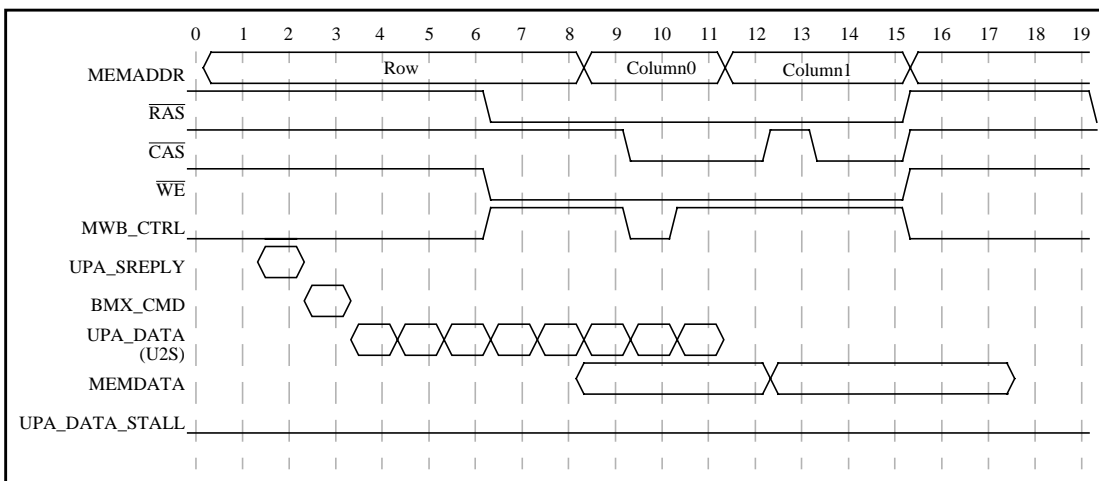


Figure 6-15 83.3-MHz U2S Write Timing

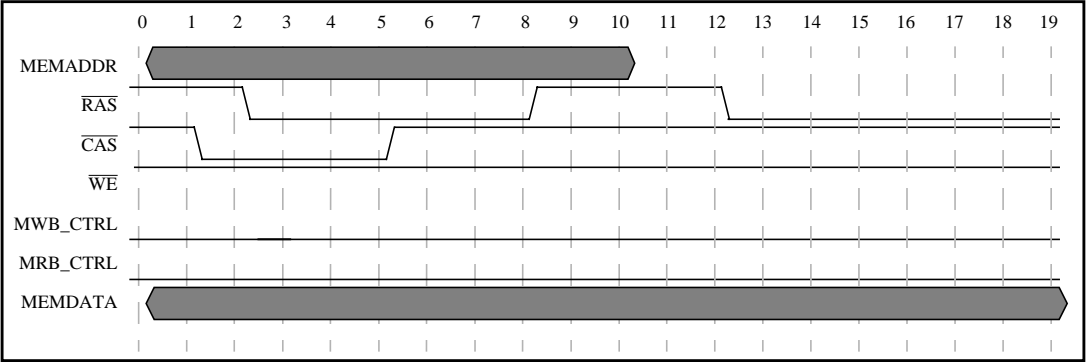


Figure 6-16 83.3-MHz Refresh Timing

6.6.3.3 71.4-MHz (14-ns) Timings

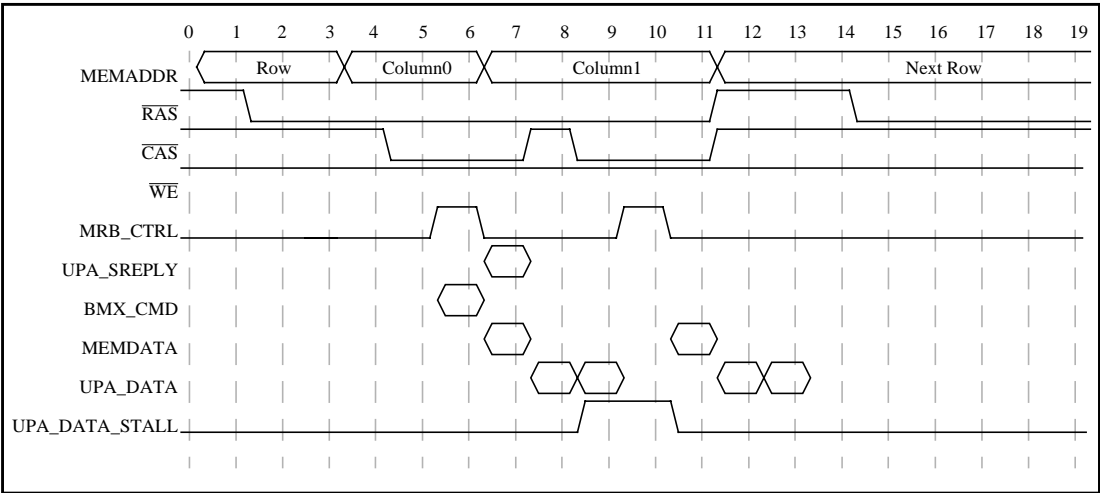


Figure 6-17 71.4-MHz CPU Read Timing

## 6. Memory System

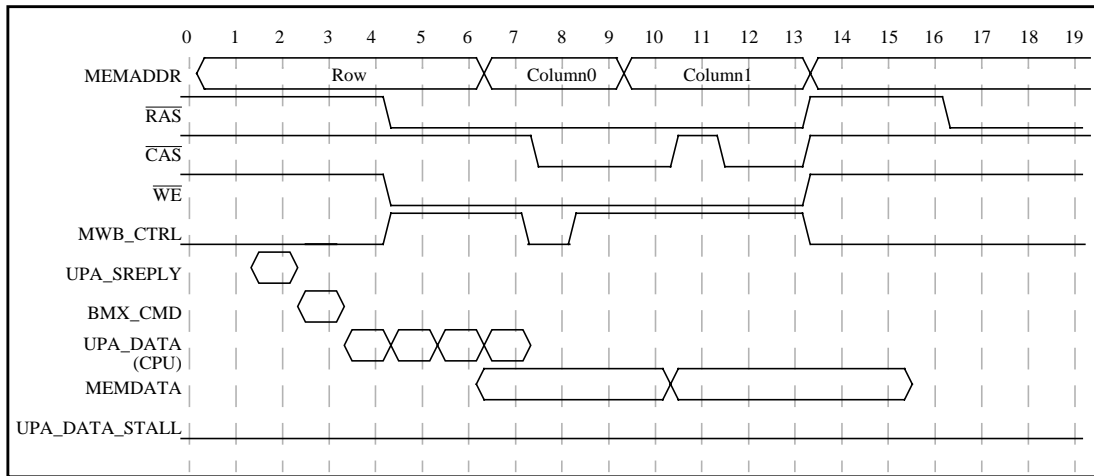


Figure 6-18 71.4-MHz CPU Write Timing

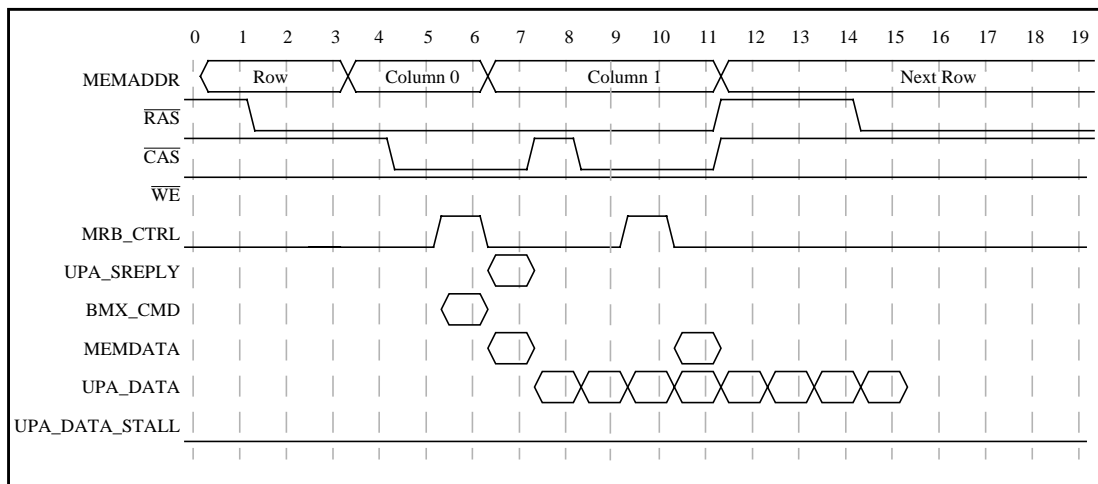


Figure 6-19 71.4-MHz U2S Read Timing

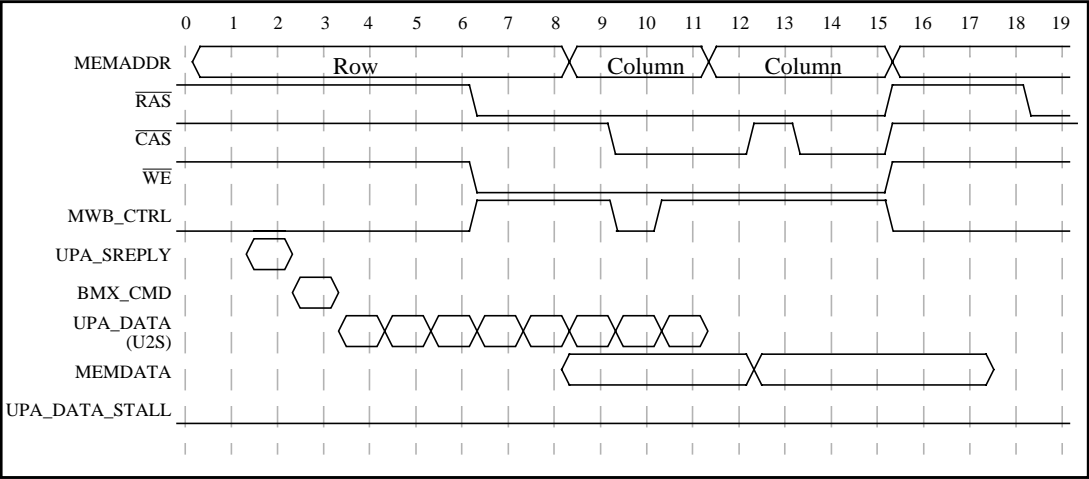


Figure 6-20 71.4-MHz U2S Write Timing

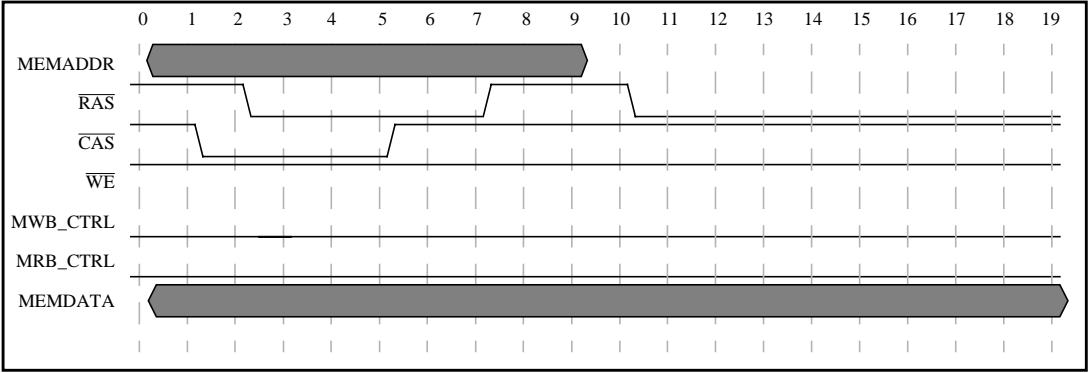


Figure 6-21 71.4-MHz Refresh Timing

### 6.6.3.4 66.7-MHz (15-ns) Timings

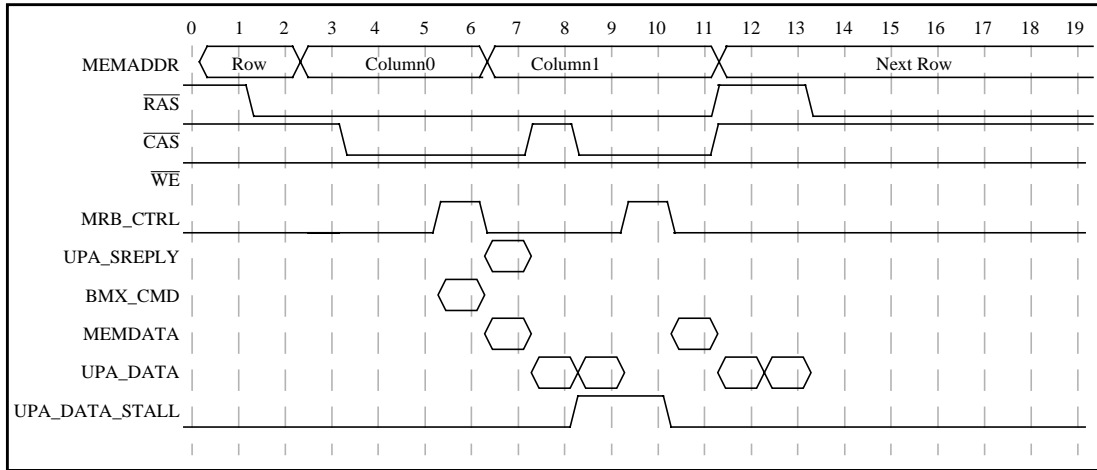


Figure 6-22 66.7-MHz CPU Read Timing

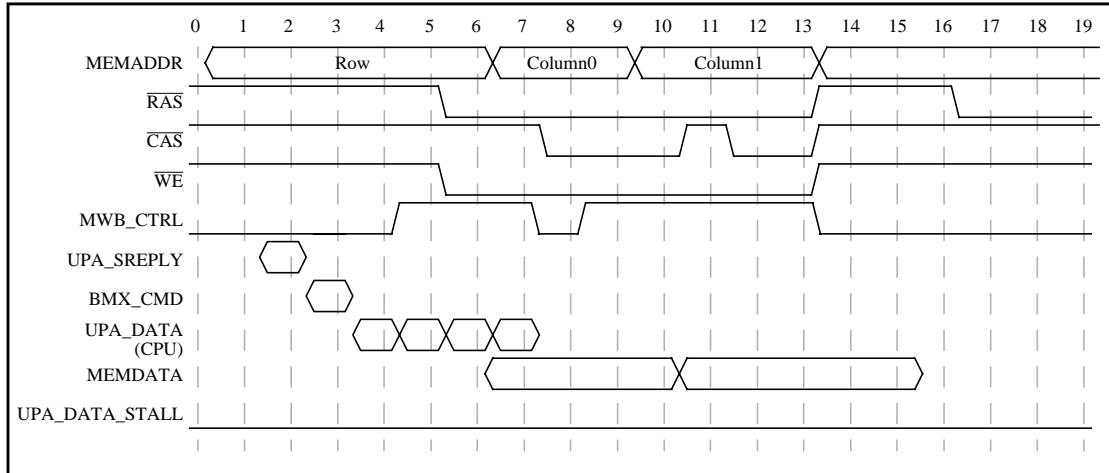


Figure 6-23 66.7-MHz CPU Write Timing

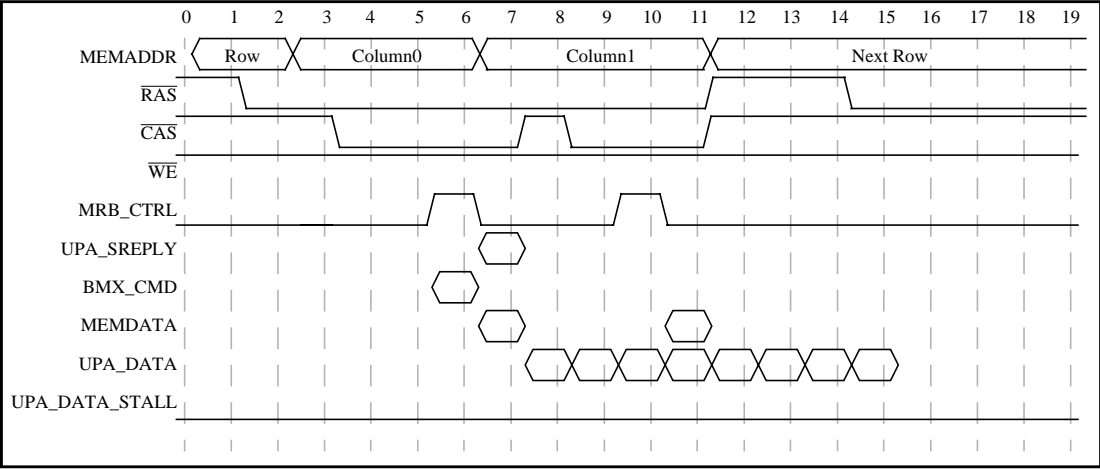


Figure 6-24 66.7-MHz U2S Read Timing

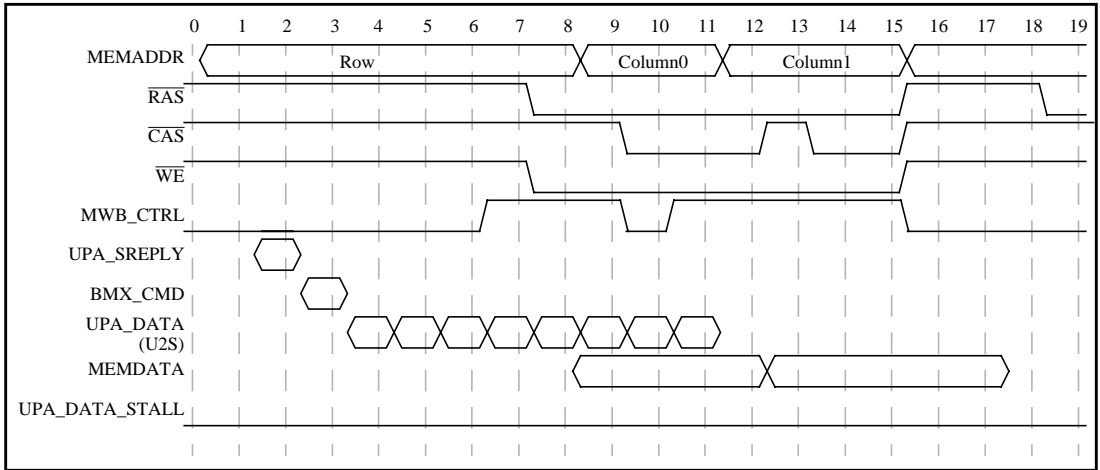


Figure 6-25 66.7-MHz U2S Write Timing



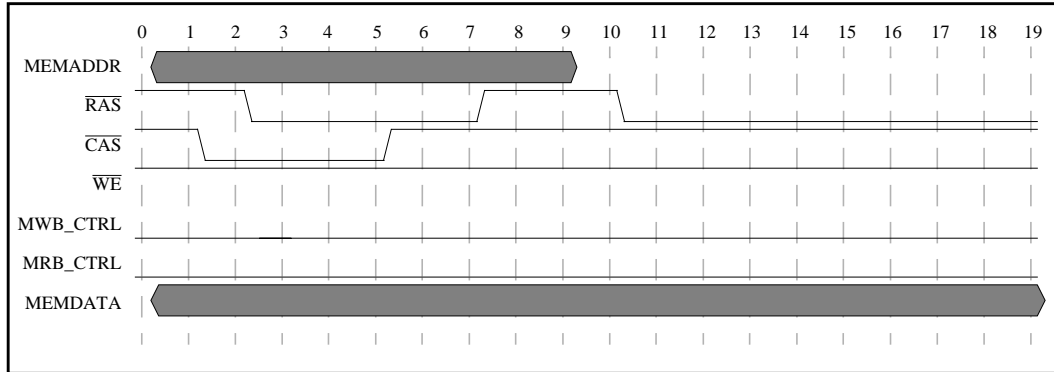


Figure 6-26 66.7-MHz Refresh Timing

### 6.6.3.5 62.5-MHz (16-ns) Timings

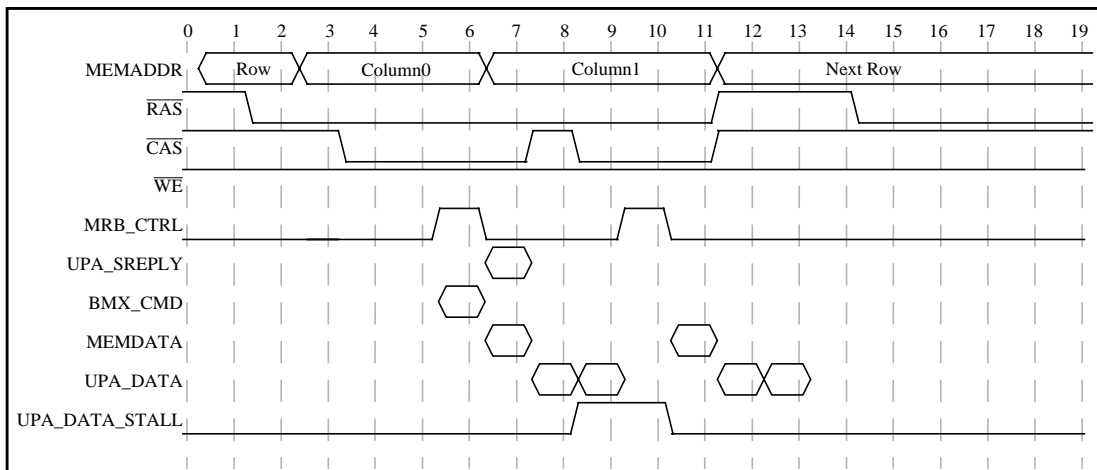


Figure 6-27 62.5-MHz CPU Read Timing

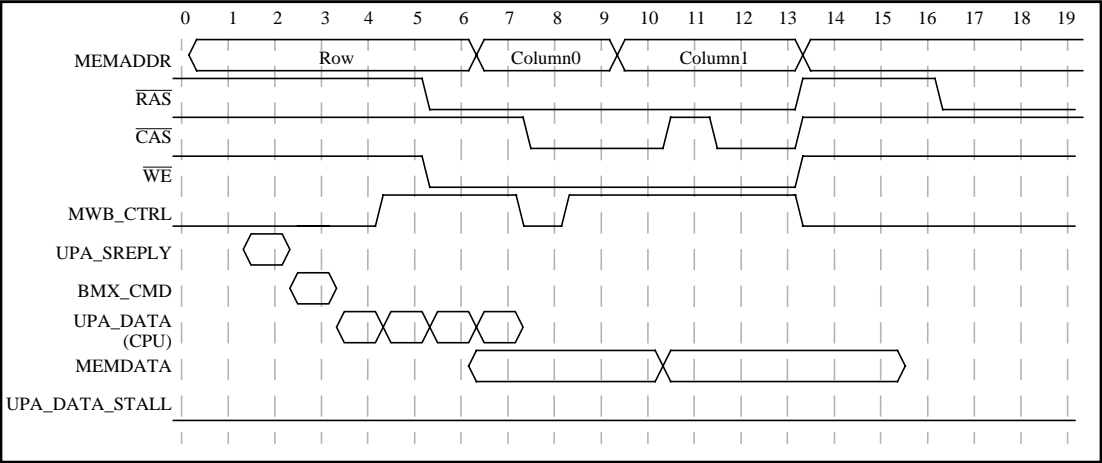


Figure 6-28 62.5-MHz CPU Write Timing

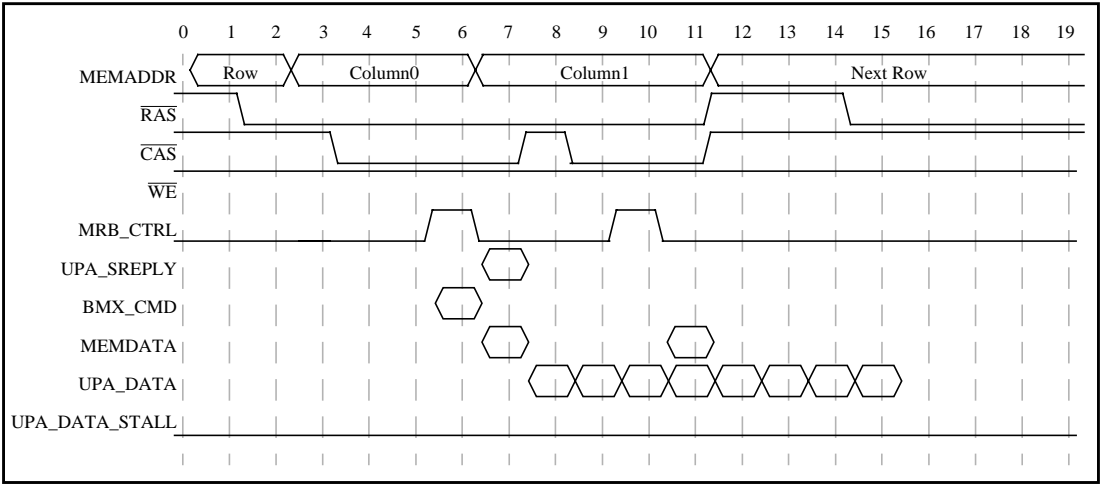


Figure 6-29 62.5-MHz U2S Read Timing

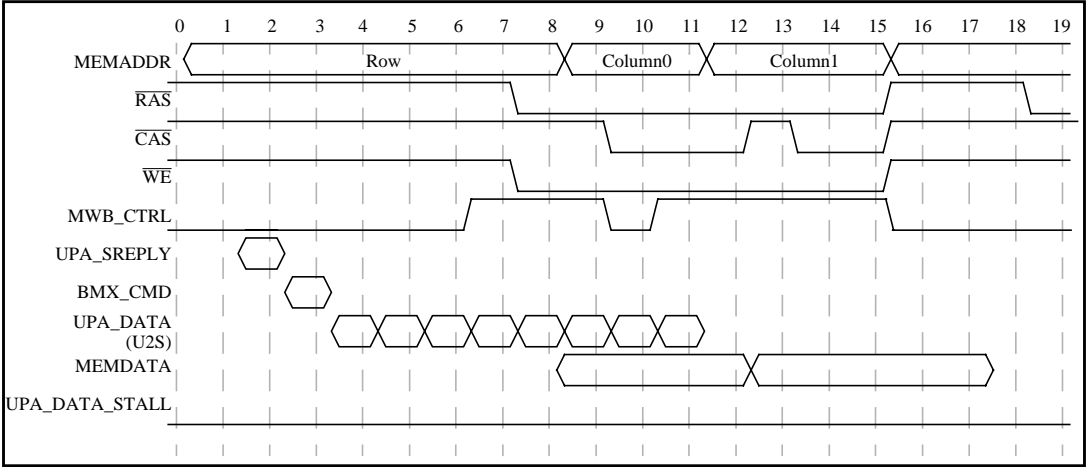


Figure 6-30 62.5-MHz U2S Write Timing

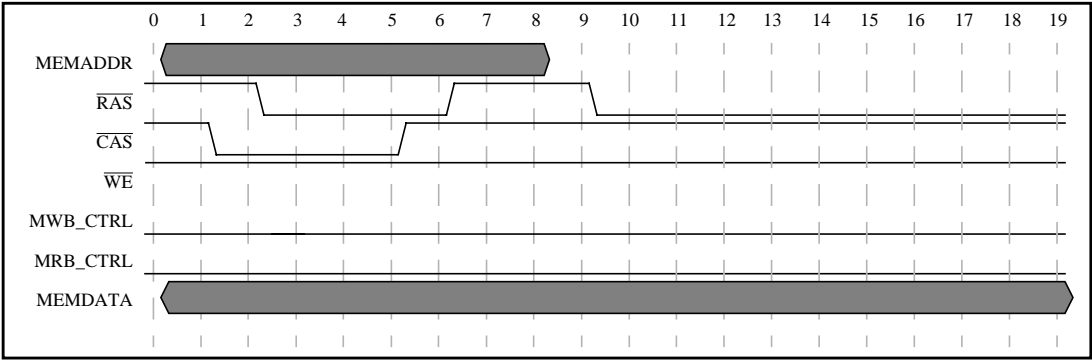


Figure 6-31 62.5-MHz Refresh Timing

6.6.3.6 55.5-MHz (18-ns) Timings

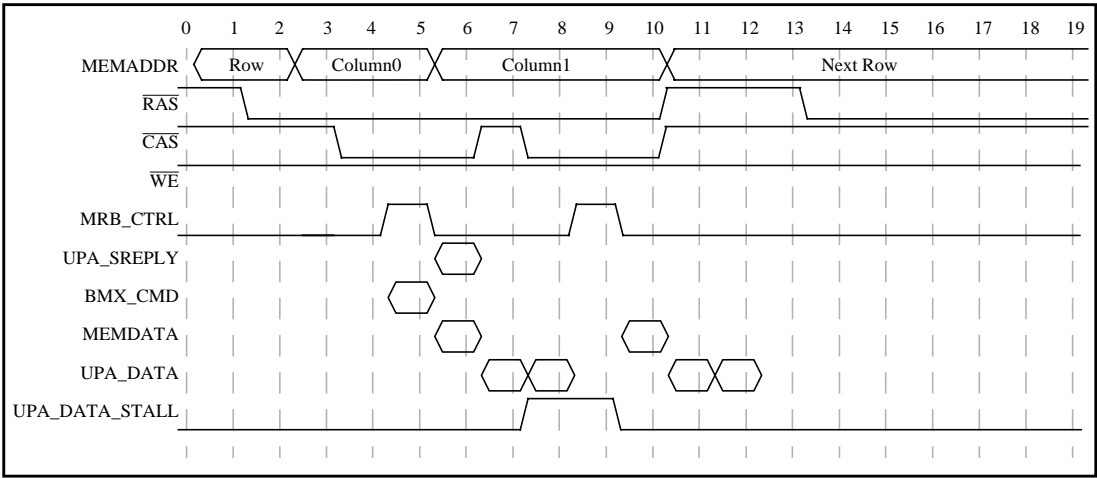


Figure 6-32 55.5-MHz CPU Read Timing

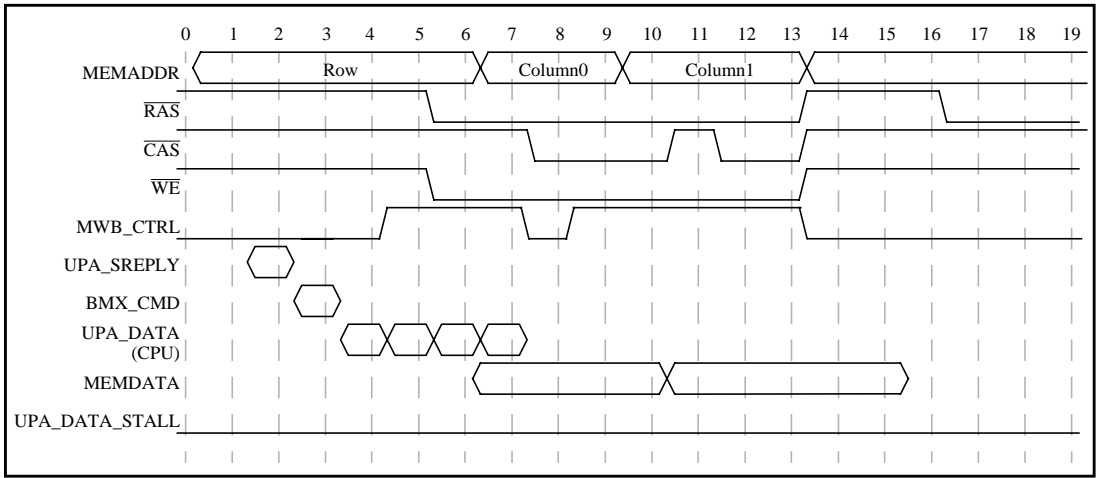


Figure 6-33 55.5-MHz CPU Write Timing

## 6. Memory System

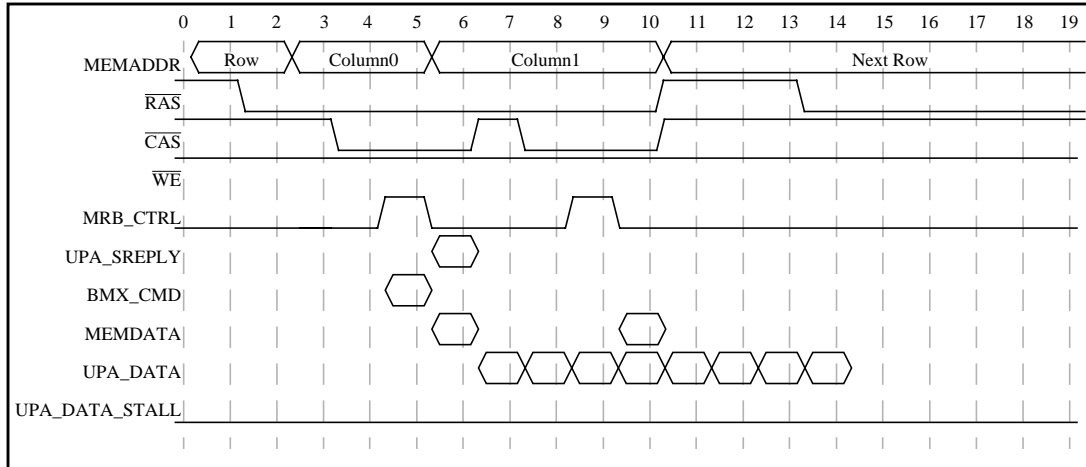


Figure 6-34 55.5-MHz U2S Read Timing

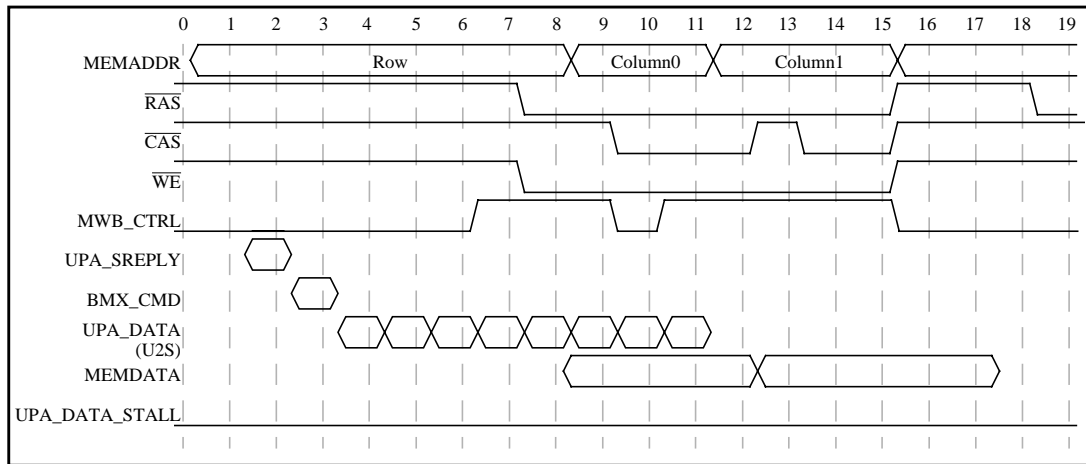


Figure 6-35 55.5-MHz U2S Write Timing

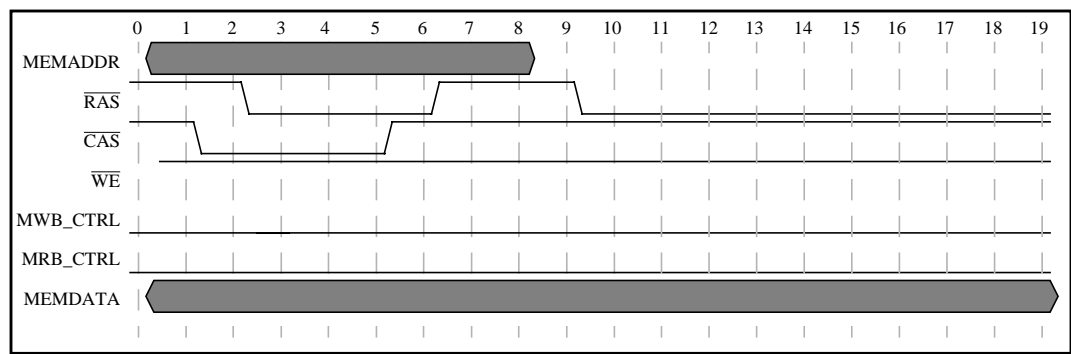


Figure 6-36 55.5-MHz Refresh Timing

6.6.3.7 50-MHz (20-ns) Timing

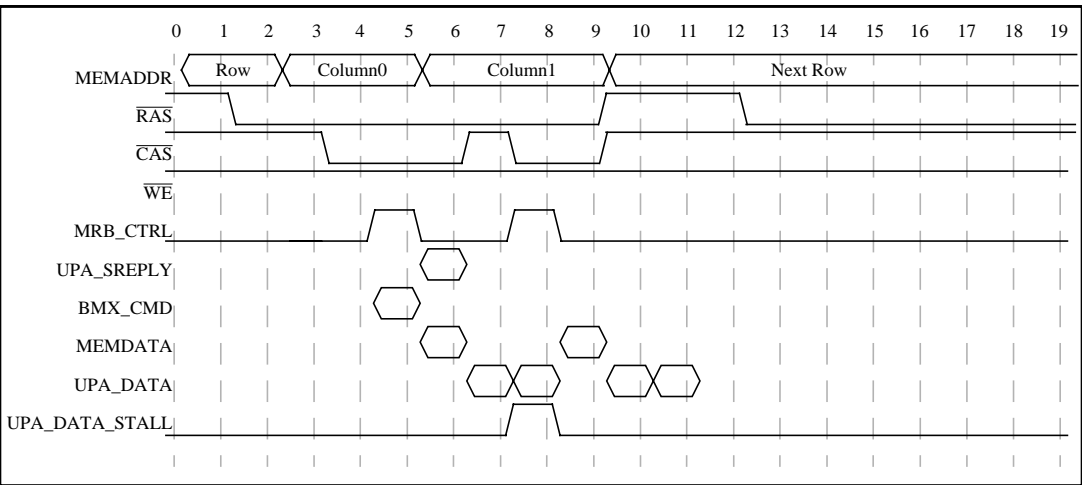


Figure 6-37 50-MHz CPU Read Timing

## 6. Memory System

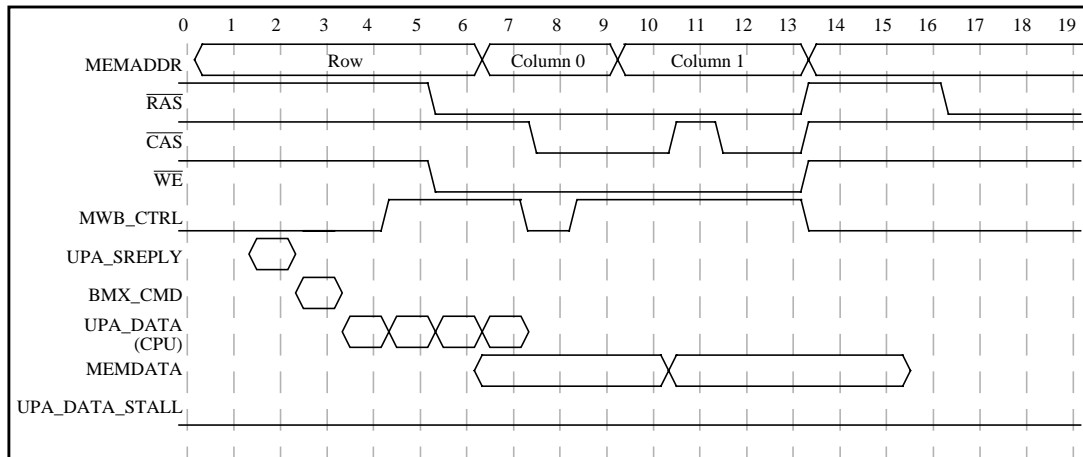


Figure 6-38 50-MHz CPU Write Timing

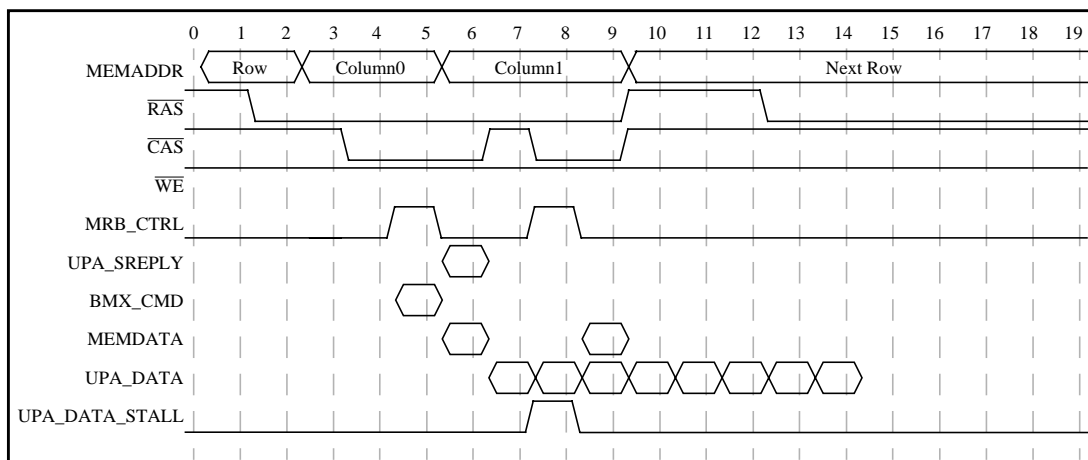


Figure 6-39 50-MHz U2S Read Timing

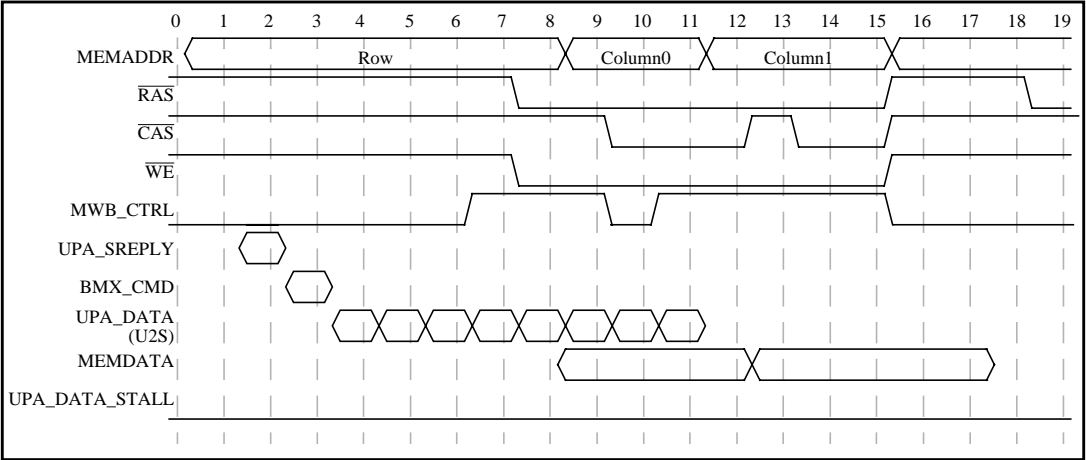


Figure 6-40 50-MHz U2S Write Timing

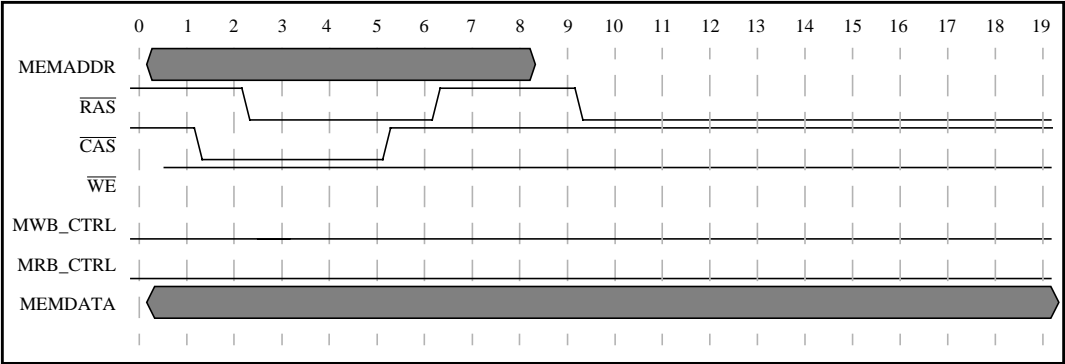


Figure 6-41 50-MHz Refresh Timing



### 6.6.3.8 Minimum Timings

Figure 6-42, "CPU Read Timing," through Figure 6-46, "Refresh Timing," show the absolute minimum timing that the memory controller is capable of generating.

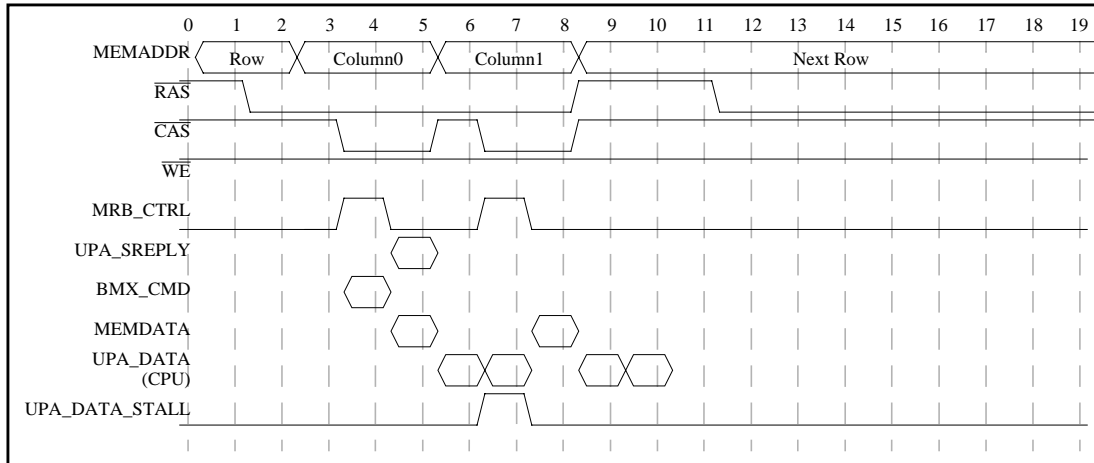


Figure 6-42 CPU Read Timing

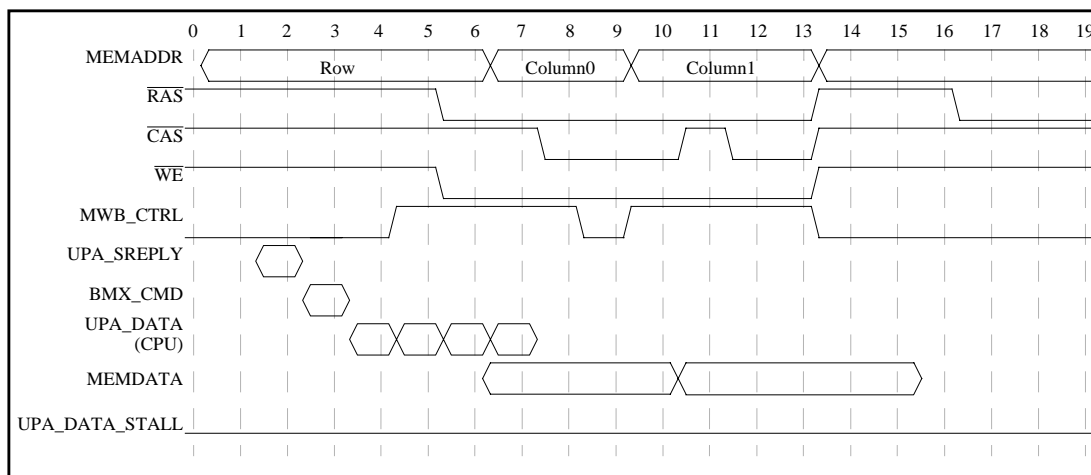


Figure 6-43 CPU Write Timing

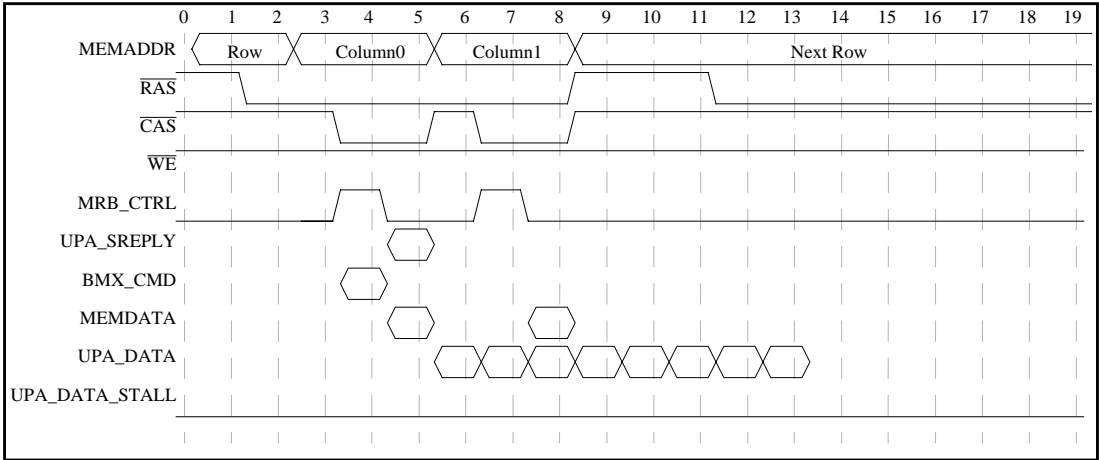


Figure 6-44 U2S Read Timing

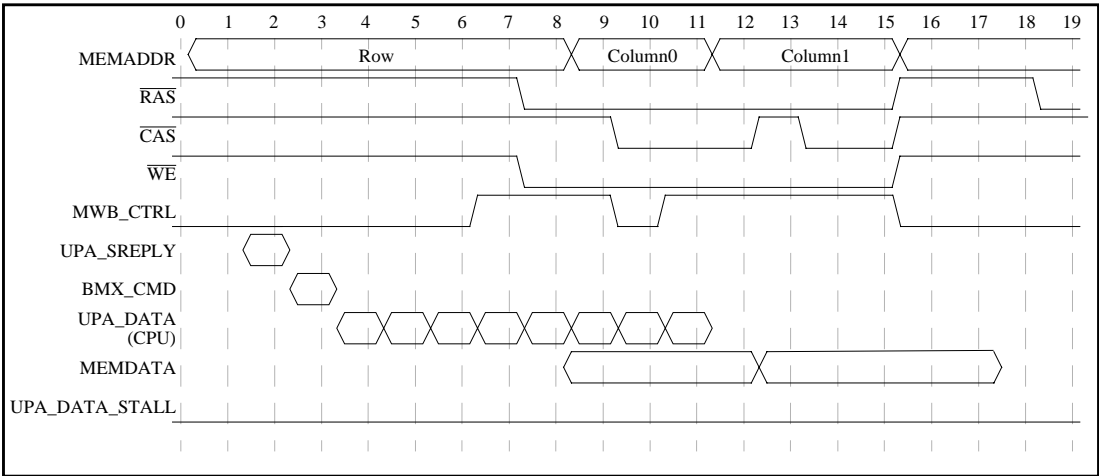


Figure 6-45 U2S Write Timing

6. Memory System

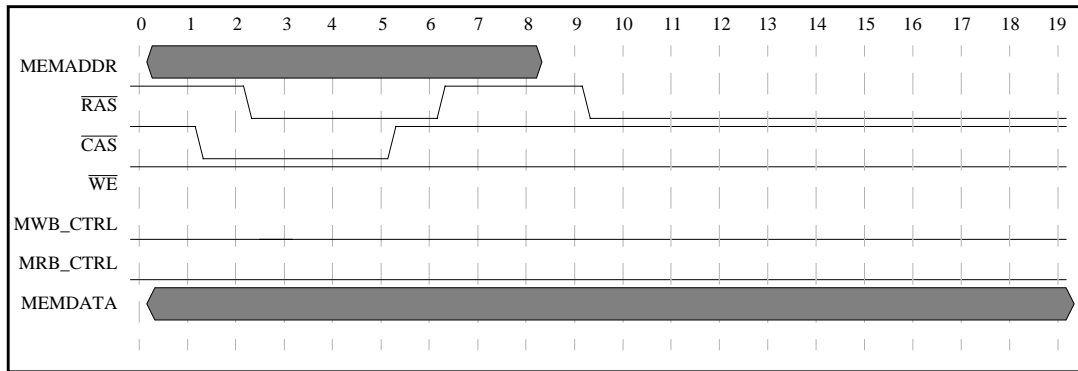


Figure 6-46 Refresh Timing



## 7.1 Introduction

All of the reset logic in the USC is contained in the EBus block. Resets may be triggered by external signals, internal register bits, or detection of internal errors. This section describes in detail what the reset behavior of the USC is. The diagram in Figure 7-1, "Resets," illustrates the reset inputs and reset outputs into and out of the USC, as well as the interconnection of resets in the system.

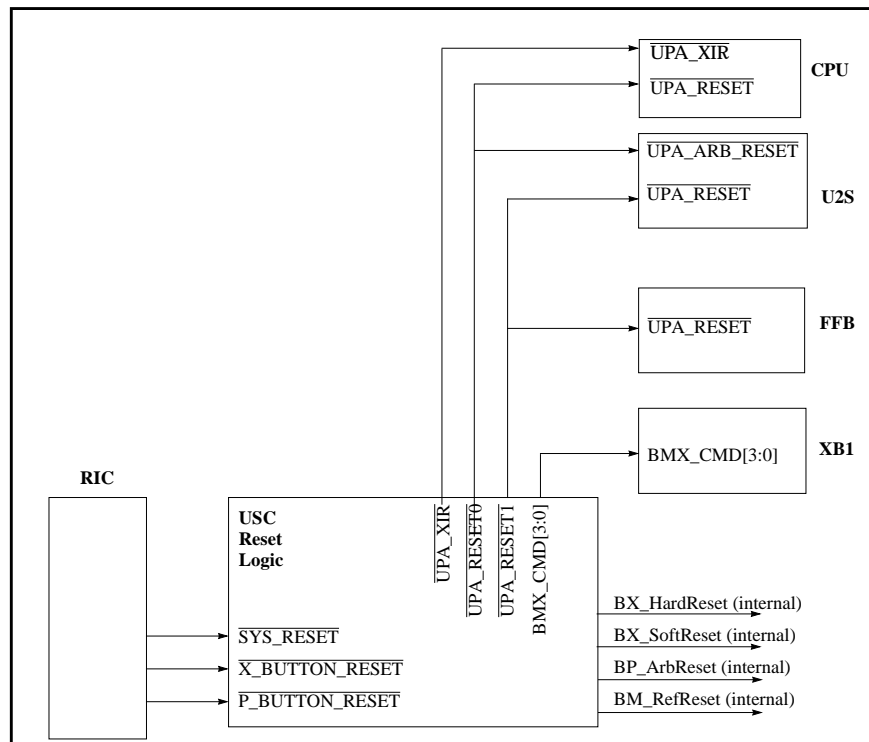


Figure 7-1 Resets

## 7.2 Reset Signals

### 7.2.1 $\overline{\text{SYS\_RESET}}$

$\overline{\text{SYS\_RESET}}$  is the power-on reset.

$\overline{\text{SYS\_RESET}}$  is supplied by the RIC, and is only asserted during power-on.  $\overline{\text{SYS\_RESET}}$  has the highest precedence, and causes everything to be reset. Because  $\overline{\text{SYS\_RESET}}$  is an asynchronous signal, it must be synchronized before being used internally by the USC.

Assertion of  $\overline{\text{SYS\_RESET}}$  causes  $\overline{\text{UPA\_RESET0}}$ ,  $\overline{\text{UPA\_RESET1}}$ ,  $\text{BX\_HardReset}$ , and  $\text{BX\_SoftReset}$  to be asserted for as long as  $\overline{\text{SYS\_RESET}}$  is asserted. After  $\overline{\text{SYS\_RESET}}$  is deasserted, the USC will keep  $\overline{\text{UPA\_RESET0}}$ ,  $\overline{\text{UPA\_RESET1}}$ ,  $\text{BX\_HardReset}$ , and  $\text{BX\_SoftReset}$  asserted for an additional 416,667 clocks, to ensure that all the internal PLLs in all of the ASICs have had sufficient time to stabilize and acquire phase lock. At 83.3 MHz, this comes out to five milliseconds.

Assertion of  $\overline{\text{SYS\_RESET}}$  will be logged by the POR bit in the SC\_Control register.

### 7.2.2 $\overline{\text{P\_RESET}}$

$\overline{\text{P\_RESET}}$  is an output of the RIC which is asserted whenever there is a  $\text{BUTTON\_POR}$  or a  $\text{SCAN\_POR}$  at the RIC. The behavior of the system in the presence of this signal is identical with  $\overline{\text{SYS\_RESET}}$  except for the logging information which indicates this as the source of the reset. This signal is synchronized to the UPA clock in the same manner as  $\overline{\text{SYS\_RESET}}$ .

### 7.2.3 $\overline{\text{X\_RESET}}$

$\overline{\text{X\_RESET}}$  is an output of the RIC which is asserted whenever there is a  $\text{BUTTON\_XIR}$  or a  $\text{SCAN\_XIR}$  at the RIC. It results in  $\overline{\text{UPA\_XIR}}$  being asserted. This signal is asynchronous to the USC, and is therefore synchronized before being used. The output generated as a result of this signal is synchronously asserted and deasserted for one clock cycle.

### 7.2.4 $\overline{\text{UPA\_RESET0}}$

$\overline{\text{UPA\_RESET0}}$  is used to reset the CPU. It is also used to reset the U2S's arbiter whenever the CPU is powered down or reset.

### 7.2.5 $\overline{\text{UPA\_RESET1}}$

$\overline{\text{UPA\_RESET1}}$  is used to reset the U2S and the FFB.

### 7.2.6 $\overline{\text{UPA\_XIR}}$

$\overline{\text{UPA\_XIR}}$  connects only to the processor. This signal is asserted for only a single UPA clock cycle.

### 7.2.7 *BX\_HardReset, BX\_SoftReset*

BX\_HardReset and BX\_SoftReset are signals generated by the EBus block and distributed internally to all logic inside the USC.

A hard reset will only occur during power-on. BX\_HardReset and BX\_SoftReset are triggered by  $\overline{\text{SYS\_RESET}}$  and causes everything to be reset.

A soft reset occurs if the USC detects a error. The cause of the error is logged in one of the USC's internal registers. BX\_SoftReset will only reset state machines and counters. It will not reset control and status registers so that the cause of the reset is preserved.

### 7.2.8 *BP\_ArbReset*

This is used to reset the UPA address bus 0 arbiter inside PIF during wakeup to ensure that the arbiter is reset to a known state.

### 7.2.9 *BM\_RefReset*

BM\_RefReset is used to reset the refresh controller in the MC. The only time BM\_RefReset is ever asserted is during a power-on reset where  $\overline{\text{SYS\_RESET}}$  is asserted. During any other type of reset, BM\_RefReset is not asserted. This is done to allow refresh to continue and the contents of memory to be preserved across a reset.

### 7.2.10 *XB1 Reset*

The XB1 is reset by asserting a BMX\_CMD of 4'b1111.

---

**Note:** The clock is required to be running in order to reset the XB1, since BMX\_CMD[3:0] is driven synchronously. As a result, all devices attached to the XB1 should make sure to have their data buses tristated during reset.

---

## 7.3 Reset Operation

The reset outputs  $\overline{\text{UPA\_RESET0}}$  and  $\overline{\text{UPA\_RESET1}}$  are asserted asynchronously and do not require the clock to be running. All outputs are driven to idle values asynchronously during reset. EBUS\_DATA[7:0] is tristated asynchronously. Outputs which contain holding amps are driven to a known idle value in order to initialize them. UPA\_ADDRBUS0[34:0] is driven to all 0, UPA\_ADR0\_PAR is driven to 1, and UPA\_ADDR0\_VAL[1:0] is driven to all 0. This means that all other clients on address bus 0 must tristate these corresponding outputs during reset to avoid a drive fight.

As described earlier, the resets will be extended by the USC for 5 milliseconds. This can cause problems for simulation, emulation, and on the tester. To get around this, there is a way to force the USC to generate a truncated reset that is extended for about eight clocks rather than 416,667. The way to enter the truncated reset mode is to assert  $\overline{\text{P\_RESET}}$  and  $\overline{\text{X\_RESET}}$  at the same time  $\overline{\text{SYS\_RESET}}$  is being asserted, and to keep them both asserted for one additional clock after  $\overline{\text{SYS\_RESET}}$  is deasserted. This situation never occurs during normal system operation.

The USC's reset strategy is summarized in Chapter 4, "Programming Model"

There are several ways in which a reset can be triggered:

1. Assertion of  $\overline{\text{SYS\_RESET}}$  input
2. Assertion of  $\overline{\text{X\_RESET}}$  input
3. Assertion of  $\overline{\text{P\_RESET}}$  input
4. Setting the SOFT\_POR bit in the SC\_Control register
5. Setting the SOFT\_XIR bit in the SC\_Control register
6. Detection of a fatal error
7. Wakeup interrupt



Table 7-1, "Reset Table," shows the same information as Table 4-7, "USC Reset Strategy," of Chapter 4, "Programming Model," but in a slightly different format. It also adds BX\_SoftReset and BX\_HardReset.

Table 7-1 Reset Table

Reset Trigger	UPA_RESET0	UPA_RESET1	UPA_XIR	BX_HardReset	BX_SoftReset	XB1 Reset	Bit Set	Note
SYS_RESET	Y	Y	N	Y	Y	Y	POR	
P_RESET	Y	Y	N	Y	Y	Y	B_POR	1
X_RESET	N	N	Y	N	N	N	B_XIR	
SOFT_POR bit	Y	Y	N	Y	Y	Y	SOFT_POR	
SOFT_XIR bit	N	N	Y	N	N	N	SOFT_XIR	
Fatal error	Y	Y	N	N	Y	Y	FATAL	2
Wakeup	Y	N	N	N	N	N	WAKEUP	3

1. The P\_BUTTON bit must be set, all other register bits must be reset.
2. Resets are generated only if the EN\_FATAL bit is set. The error is logged regardless of whether EN\_FATAL is set or not.
3. UPA Address Bus 0 arbiter must be reset.

## 7.4 Sleep and Wakeup

Only the processor is allowed to go to sleep in the reference platform system. The processor has to set the SLEEP bit in its own SC\_Port\_Config register. After the SLEEP bit is set, whenever an interrupt packet is directed to the processor, the USC will issue an S\_INAK, and the interrupter is forced to resend the interrupt. Any reads directed to the sleeping processor will cause an S\_ERR to be issued, and any writes will be dropped on the floor, per the UPA Interconnect Architecture document.

The processor should then set the EN\_WAKEUP\_POR bit in the SC\_Control register. After this bit is set, whenever a wakeup event occurs (a valid interrupt from the U2S directed towards the processor), the USC will first assert UPA\_SC\_REQ0 to gain ownership of address bus 0. This is done to prevent a truncated packet from being sent on the bus. After that, it will assert UPA\_RESET0 to wake up the processor and to reset the U2S, UPA arbiter. The wakeup event will also clear the EN\_WAKEUP\_POR bit, so that subsequent wakeup event will not trigger more assertions of UPA\_RESET0 from the USC.

Until the processor wakes up and clears its SLEEP bit, the USC will continue sending S\_INAK to the U2S. When the SLEEP bit is cleared, the USC will forward the interrupt the U2S.



### *8.1 Introduction*

Since the USC does not contain a data path, an eight-bit EBus port has been provided to allow reading and writing of the USC's internal registers.

The EBus is controlled by the STP2001 (SLAVIO). The SLAVIO can handle up to three EBus clients: EPROM, TOD/NVRAM, and a generic port. The USC is hooked up to the generic port. Therefore, the USC's register space is actually part of the SBus address space.

The EBus interface is implemented in the EBus block (EB).

### *8.2 Functional Description*

Since the EBus runs asynchronously with respect to the USC's clock - EBus signals are clocked using the SBus clock - all EBus inputs to the USC pass through a boundary register and a dual-ranked synchronizer. All outputs from the USC to the EBus (EB\_RDY and EB\_DATA) are clocked out through a boundary register using the USC's clock and are synchronized by the SLAVIO using dual-ranked synchronizers.

Registers in the USC are read or written using one level of indirection. The EB contains only two registers, an 8-bit EB\_ADDR register and a 32-bit EB\_DATA register. Whenever the processor wants to read or write to one of the USC's internal registers, it must first write the internal address of the register it wants to access into the EB\_ADDR register, then perform a read or write to the data register. The address map for the USC's internal registers is found in Table 4-1 of Chapter 4, "Programming Model."

Table 8-1 EBus Register Map

EB_ADDR[2:0]	Register
000	EB_ADDR[7:0]
001	Maps into EB_ADDR[7:0]
010	Maps into EB_ADDR[7:0]
011	Maps into EB_ADDR[7:0]
100	EB_DATA[31:24]
101	EB_DATA[23:16]
110	EB_DATA[15:8]
111	EB_DATA[7:0]

Only three bits are needed for EB\_ADDR.

There are some very strict rules that the processor must follow when it is accessing the USC through the EBus, since it only supports byte reads and writes, and is attempting to access 32-bit data.

For reads, the 32-bit datum pointed to by the contents of the EB\_ADDR register is fetched into a 32-bit staging register when byte 0 (EB\_ADDR[2:0] == 100) is read, and byte 0 is returned on the EBus. Successive reads of bytes 1, 2, and 3 of the EB\_DATA register will return the remaining three bytes in the staging register.

For writes, the processor must begin by writing to byte 0 of the EB\_DATA register, then bytes 1, 2, and 3. The EBus block will store the bytes into a 32-bit staging register until all four bytes have been written into it. The EBus block will not actually update the register pointed to by the contents of EB\_ADDR until byte 3 is written.

The read or write should be done as an atomic operation. Atomicity is not enforced in hardware and has to be enforced in software using some sort of lock. EBus accesses should never be interrupted, and only one processor can be allowed access to the EBus at any given time. Otherwise results will be unpredictable and incorrect.

Internally the EB will decode the address and generate a select for either the PIF, MC, or CC (for MP only), and pass on the register address. It will perform all the packing and unpacking and handshaking necessary to complete the transfer.

The  $\overline{\text{EB\_RDY}}$  signal is asserted by the USC as a handshake to control data flow. The USC will keep  $\overline{\text{EB\_RDY}}$  asserted until it detects that  $\overline{\text{EBUS\_RD}}$  or  $\overline{\text{EBUS\_WR}}$  has been deasserted.

During reads, the USC will begin driving  $\text{EBUS\_DATA}[7:0]$  as soon as it detects that  $\overline{\text{EBUS\_RD}}$  is asserted. Data will not be valid for several clocks while the fetch is done internally. Then  $\overline{\text{EBUS\_RDY}}$  is asserted.  $\text{EBUS\_DATA}[7:0]$  is tristated and  $\overline{\text{EBUS\_RDY}}$  is deasserted when the USC detects that  $\overline{\text{EBUS\_RD}}$  has been deasserted.

### 8.3 Timing Diagrams

Figure 8-1, "Internal Read Timing," through Figure 8-4, "External EBus Write Timing," show both internal timing and external timing for EBus accesses.

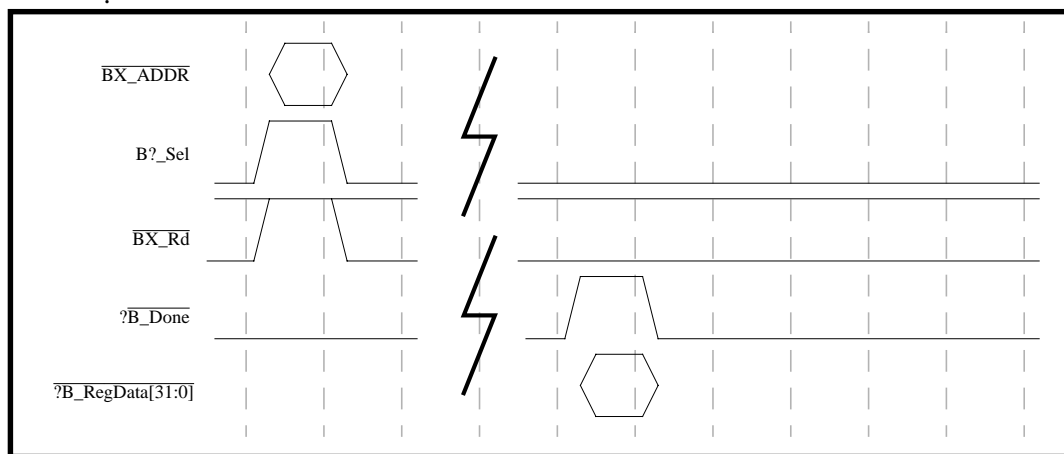


Figure 8-1 Internal Read Timing

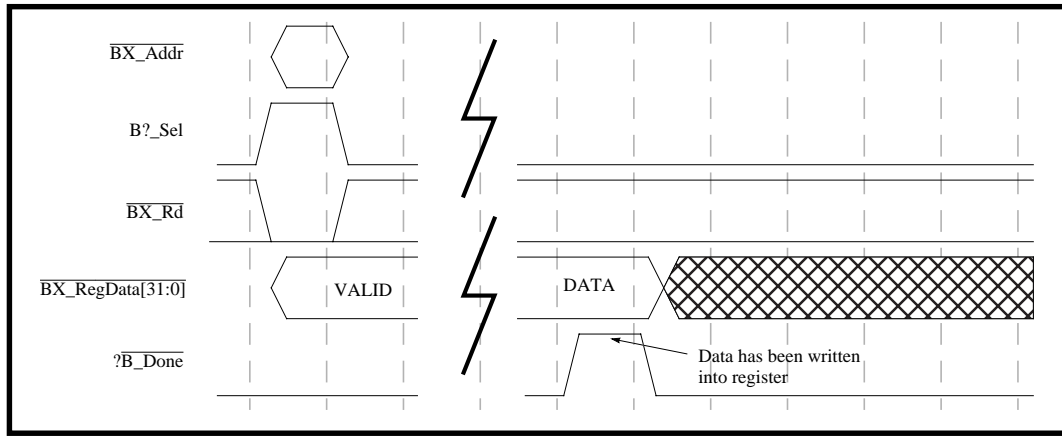


Figure 8-2 Internal Write Timing

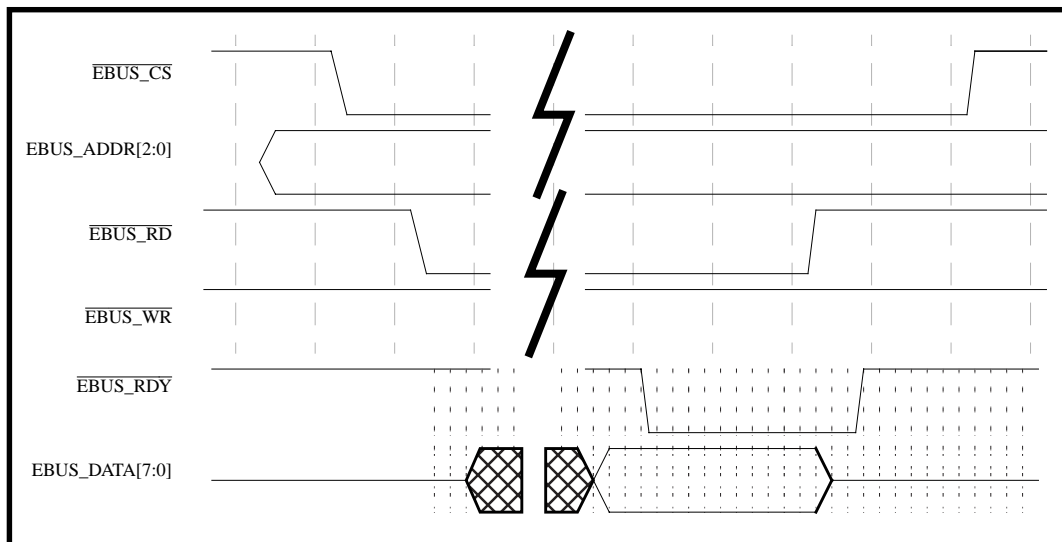


Figure 8-3 External EBus Read Timing

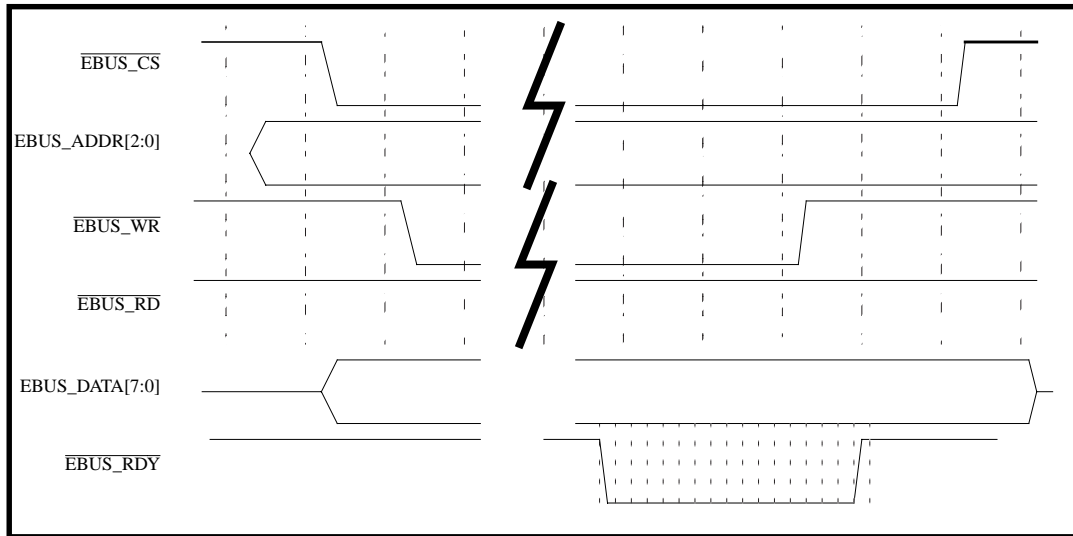


Figure 8-4 External EBus Write Timing





### 9.1 Introduction

This section contains topics that do not fit well into any of the other sections, but yet do not warrant a section of their own. It describes some secondary functions and features of the chip that are not absolutely essential to the basic functioning of the chip.

### 9.2 Process Monitor

The USC contains a “process monitor” which can be used to measure delays inside the chip. A schematic is shown in Figure 9-1, “Process Monitor.”

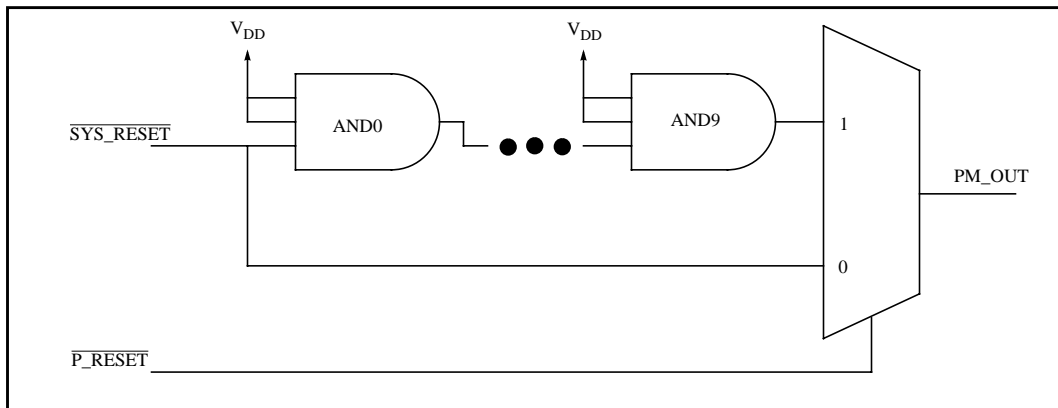


Figure 9-1 Process Monitor

The circuit consists of two different paths going into a 2-to-1 multiplexer. One path goes directly to one input of the multiplexer, while the other path goes through a chain of thirty 3-input *AND* gates, then into the other input of the multiplexer.

Due to the lack of spare pins, some of the existing input pins are multiplexed to supply inputs to the process monitor circuit. The  $\overline{P\_RESET}$  input is used to select between the short and long path.  $\overline{SYS\_RESET}$  is the input to be measured against. The output of the process monitor goes to a dedicated output pin, PM\_OUT.

### 9.3 Spare Gate Macros

There are a total of 34 spare gate macros scattered inside the USC. The SGMP module is used, which is described in the ATT HL400C CMOS databook. Their locations in the RTL can be found from the Verilog code.

### 9.4 Debug Pins

The USC contains four debug pins, DEBUG[3:0], which allow the user to observe various internal states. The debug logic all resides in the EB block, and consists of a four-bit wide 16-to-1 multiplexer, along with a 16-bit control register (SC\_Debug). By setting the appropriate bit in the SC\_Debug register (one hot encoding is used), the user can select one of 16 sets of internal state to be visible on the DEBUG pins.

The signals are all registered before going into the multiplexer, then registered again before being sent out to avoid timing violations. The user needs to be aware of this two clock delay.

The debug pin information is described in Chapter 3 of the reference platform system documentation.

### 9.5 Performance Counters

The performance counters are documented in the reference platform system document. There are two 32-bit counters plus one control register. Since it is often necessary to compare the contents of one counter to the other counter, there is a shadow register that will get loaded with the contents of one counter when the other counter gets read. This freezes the contents of the one counter so that it will not continue to increment while the other counter is being read, preventing inaccurate results.

## *9. Miscellaneous Items*



## ***Index***

---



### **A**

- Address Bus 1 64
- address buses 14
- address partitioning 21
- Address port 0 1
- Address port 1 1
- addressing 76
- ATPG 2
- B**
- BGA 3
- Block Diagram 74
- Blocking 55
- Blocking Conditions 55, 56
- BM\_RefReset 109
- BP\_ArbReset 109
- BX\_HardReset 109
- BX\_SoftReset 109
- byte-access register 23

### **C**

- C\_Control register 16
- Cached Transactions 56
- cancel bi 49
- Cancelled Writebacks 49
- class symmetr 58
- Coherence Algorithm 48

- coherent read 51

- Coherent Read from Processo 50

- coherent read transaction 49

- Coherent Requests 52

- Coherent requests 48

- coherent requests 60

- coherent transaction 48

- Coherent transactions 48

- coherent transactions 70

- Coherent Write 52

- Coherent writes 51

- Concurrency 61

- control register 120

- Crossbar Switch 2

### **D**

- Data Path Control 2

- Data Path Scheduler 13, 15

- data path scheduler 61

- data stall 59

- debug logic 120

- debug pin 16

- Debug Pins 120

- Default Memory Timing 85

- default timing 85

- DPS 15, 50, 51, 52, 53, 56, 61

- DSIMM 73
- DSIMMs 2
- E
  - EB 16, 113, 114
  - EB bloc 120
  - EB block 16
  - EBus 43, 107, 113
  - EBus accesses 115
  - EBus block 113, 114
  - Ebus Block 109
  - EBus Interface 2, 13, 16, 113
  - EBus port 113
  - ECC 74
  - EPROM 113
- F
  - Fast path 70
  - Fast-path timing 82
  - FFB 52, 67, 109
  - Flow Control 54
- G
  - generic port 113
  - global registers 16
- I
  - Interface Signals 8
  - internal address map 21
  - internal registers 24, 113
  - interrupt 54
  - IVA bit se 50
- J
- JTAG 2
- M
  - Master ID Assignment 13
  - MC 15, 50, 51, 61, 74, 77, 114
  - MC timing 80
  - MC\_Control0 15
  - MC\_Control0 Register 38
  - MC\_Control1 15
  - Memory Access Timings 83
  - Memory Control Register 40
  - Memory Controller 13, 15
    - memory controller 2, 14, 56, 61, 74, 76
    - memory data bus 74
  - Memory Interface 2
    - memory reads 83
    - memory request 51
    - memory request packet 82
    - memory system 73, 74
    - memory writes 83
  - multiplexer 120
  - muxing logic 77
- N
  - nominal timing 69
  - Noncached Block Read 65
  - Noncached Reads 53
  - Noncached Single Write 68
  - Noncached transaction 52
  - noncached transaction 55
  - Noncached Writes 53
  - Nonfatal errors 62
- P
  - P\_Reply 52, 62, 65, 67
  - P\_RESET 108, 120
  - P0\_Config Register 34
  - Packet Handling 45
  - Performance Counter Regist 31
  - Performance Counters 120
  - Performance counters 16
  - performance registers 32
  - PIF 14, 48, 49, 50, 51, 52, 53, 60, 61, 114
  - PIF block 60
  - Pin Count 12
  - PLL 3
  - PM\_OUT 120
  - Port Interface 13
  - presence detection 58
  - Process Monitor 119

- proper slave 14
- R
- read buffer 51
- reads 115
- register address 114
- register programming 84
- register space 113
- register values 84
- Registers 21, 113
- registers 25
- Requests 58
- requests 61
- reset 110
- reset bits 27
- reset inputs 107
- reset logic 107
- reset output 107
- reset outputs 110
- Resets 2, 107
- RIC 2, 108
- RMW 49
- RMW flag 49
- RTL 120
- RTL Directories 18
- S
- S\_Replys 55, 56
- SBus address 113
- SBus clock 113
- SC\_Address Register 22, 23, 24
- SC\_Control Register 26
- SC\_Data Register 22
- SC\_Data Register 23, 24
- SC\_Debug register 16
- SC\_ID Register 30
- SC\_ID register 16
- SC\_Perf\_Ctrl Register 31
- SC\_Perf\_Ctrl register 30
- SC\_Perf0 31
- SC\_Perf0 Register 30
- SC\_Perf1 Register 31
- SC\_PerfShadow Register 31
- SC\_Port\_Config registers 15, 54
- SC\_Port\_Status registers 15
- SC\_UP. 1
- scoreboard 60
- SGMP module 120
- SIMM 75
- SIMMs 73, 74
- slave request 60
- SLAVIO 2, 16, 22, 113
- sleep 111
- soft reset 109
- software-visible register 21
- Source and Destination Designators 17
- SPARCstation 20 73
- Spare Gate Macros 120
- SRequest 49, 50, 51, 52, 55, 63
- Status Register 24
- STP2001 113
- SYS 108
- SYS\_RESET 108, 120
- SYSIO\_Status Register 37
- System Controller 5
- system controller 21
- T
- Timing 100
- Timings 88, 90, 93, 95, 98, 103
- TOD/NVRA 113
- U
- U2S 14, 48, 49, 51, 52, 54, 55, 58, 70, 83, 108, 111
- UltraSPARC Port Architecture 1
- UltraSPARC™ - I Reference Platform 1
- UltraSPARC-1 58
- UltraSPARC-I 50, 53
- UltraSPARC-I Reference Platform 5

- uniprocessor system 1
- Uniprocessor System Controller 1
- uniprocessor system controller 14
- UPA 1, 8, 13, 82, 109
- UPA Address Bus 0 60
- UPA arbiter 111
- UPA Architecture 62
- UPA clock 108
- UPA clock cycle 109
- UPA Interface Signals 8
- UPA Output 61
- UPA packets 45
- UPA Port Interface 14
- UPA Port Registers 24
- UPA Ports 1
- UPA\_RESET0 108
- UPA\_RESET1 109
- UPA\_XIR 109
- UPA64 65
- UPA64S 1, 14
- UPA64S client 58
- UPA-to-SBus 37
- UPA-to-SBus Interface 36
- USC 2, 7, 13, 21, 22, 24, 25, 30, 48, 50, 52, 54, 55, 58, 62, 64, 74, 80, 82, 107, 108, 110, 113, 115, 120
- USC Indirect Addressing 24
- USC Internal Register 24
- USC Memory Controller 38
- USC Memory Controller Registers 24
- USC Overall Control 24
- USC Port Interface Registers 34
- USC Queues 14
- USC Register Notation Convention 24
- USC reset strategy 27
- USC System Addresses 22
- USC Transaction Table 45
- USC's clock 113
- USC's internal registers 25
- V
- Verilog 13, 18, 120
- W
- wakeup 111
- WRI 50
- X
- X\_RESET 108
- XB1 2, 50, 52, 56, 77, 83
- XB1 Crossbar Switc 15
- XB1 read buffer 50
- XB1 Reset 109