TECHNICAL MANUAL

L64364
ATMizer® II+
ATM-SAR Chip

February 2001

LSI LOGIC®

R14008.A

Document DB14-000037-02, Second Edition (February 2001)
This document describes LSI Logic Corporation's L64364 ATMizer® II+ ATM-SAR Chip and will remain the official reference source for all revisions of this product until rescinded by an update.

To receive product literature, visit us at http://www.lsilogic.com.

TRADEMARK ACKNOWLEDGMENT

BD

Preface

This book is the primary reference and technical manual for the L64364 ATMizer® II+ ATM-SAR Chip. It contains a complete functional description of the L64364 and includes complete physical and electrical specifications for the L64364.

Audience

This book assumes that you have some familiarity with the concepts of Asynchronous Transfer Mode (ATM), data communications, microprocessors, and related support devices. The people who benefit from this book are:

- Engineers and managers who are evaluating the L64364 for possible use in ATM applications.
- Engineers who are designing the L64364 into a system.
- Software developers writing software for the L64364.

Organization

This book has the following chapters:

- Chapter 1, **Introduction**, provides an overview of the ATMizer II+ ATM-SAR Chip and lists its features.
- Chapter 2, **Functional Overview**, describes the ATMizer II+ chip on a functional block level.
- Chapter 3, **Signal Descriptions**, lists and describes all of the input/output signals of the ATMizer II+ chip.
- Chapter 4, **ATM Processing Unit**, describes the architecture, instruction set, registers, cache memory, memory map, interrupts, exceptions, and boot procedures for the ATM Processing Unit.

- Chapter 5, **Enhanced DMA**, describes the data structures used by the EDMA, the EDMA commands, its registers, and its operation in AAL0 and AAL5 modes.
- Chapter 6, **ATM Cell Interface**, describes the ATM cell size and layout, the cell descriptor, registers, Cell Buffer Manager, receiver, transmitter, Utopia polling schemes, and loopback mode for the ATM Cell Interface.
- Chapter 7, **Scheduler Unit**, describes the Scheduler Unit's modes of operation, command execution, and registers.
- Chapter 8, **Timer Unit**, describes how the timer clocks are selected, the timer registers, and time-out events.
- Chapter 9, **PCI Interface**, describes the PCI registers, master and slave transactions, and how the ATMizer II+ chip balances bus usage.
- Chapter 10, **Secondary Bus Memory Controller**, describes the controller configuration, discusses bus performance considerations, and describes the operation of the individual controllers.
- Chapter 11, **System Clock**, discusses clock selection and describes its synthesis using a phase-locked loop.
- Chapter 12, **JTAG Interface**, describes the JTAG instructions supported by the L64364 and the bit order of the L64364 boundary scan chain.
- Chapter 13, **Specifications**, provides AC timing figures, electrical requirements, pinout information, and package information for the ATMizer II+ chip.
- Appendix A, **Register Summary**, provides a brief summary of all of the registers in the ATMizer II+ chip and includes page number references for their descriptions.
- Appendix B, **The ATM Cell**, describes the layout and fields in the ATM cell header and the AAL5 trailer.
- Appendix C, **Glossary of Abbreviations**, lists the abbreviations used in the manual and defines them.

Related Publications

LSI Logic's *MiniRISC® CW4011 Superscalar Microprocessor Core Technical Manual,* Order No. C14040

ATM Forum - Utopia Level 1 and Level 2, V1.0, af-phy-0039.00

PCI Local Bus Specification 2.1

IEEE 1149.1, Standard Test Access Port and Boundary Scan Architecture

Conventions Used in This Manual

The following signal naming conventions are used throughout this manual:

- Signal names are in uppercase characters. Active-LOW signals have a lowercase "n" at the end of the signal name (for example, RESETn) while active-HIGH signals do not.
- Multiple control signals such as external interrupts are grouped (for example, EXT_INTn[5:0]), but they are treated as control signals.
- Signal names, commands, and register bits and fields are courier.
- All CW4011 core and CW4011 shell interface signals are unidirectional.

The word *assert* means to drive a signal true or active. The word *deassert* means to drive a signal false or inactive. The word *set* means to change a bit (in registers, descriptors, etc.) from logical 0 to logical 1. The word *clear* means to change a bit from 1 to 0.

Hexadecimal numbers are indicated by the prefix "0x"—for example, 0x32CF. Binary numbers are indicated by the prefix "0b"—for example, 0b0011.0010.1100.1111. The hexadecimal form is used as much as possible for all numbers with four or more bits.

The L64364 requires over 100 internal and external registers. These are described in the appropriate chapters and sections. Register references in other sections of the manual are followed by the page number of their description in parenthesis. In addition, Appendix A lists the registers by functional area, includes their addresses, and provides page number references to their descriptions.

Contents

Chapter 1 Introduction

1.1 Overview 1-1
1.2 Functional Description 1-2
1.3 Features 1-3

Chapter 2 Functional Overview

2.1 Major Functional Units 2-1
2.2 ATM Processing Unit (APU) 2-3
2.3 Enhanced DMA (EDMA) 2-3
2.4 ATM Cell Interface (ACI) 2-4
2.5 Scheduler and Timer Units 2-5
2.6 PCI Interface 2-5
2.7 Endian Considerations 2-6
2.8 Secondary Bus Memory Controller (SBC) 2-7
2.9 Other Features 2-7

Chapter 3 Signal Descriptions

3.1 I/O Signals Summary 3-1
3.2 PCI Interface 3-4
3.3 Secondary Memory Interface 3-8
3.4 Utopia Interface 3-10
3.5 Clocks and Utility Signals 3-15
3.6 APU Signals 3-17
3.7 Serial EPROM Interface 3-18
3.8 JTAG Test Interface 3-19
3.9 Power and Ground Pins 3-20

Chapter 4 ATM Processing Unit

4.1 APU Overview 4-1
4.1.1 Block Diagram 4-2
4.1.2 Features 4-3
4.2 APU Architecture 4-3
4.2.1 CW4011 Core 4-4
4.2.2 Cache and External Interface 4-6
4.2.3 CW4011 Pipeline 4-6
4.3 APU Instruction Set Summary 4-10
4.3.1 Instruction Set Formats 4-13
4.3.2 Load and Store Instructions 4-14
4.3.3 Computational Instructions 4-17
4.3.4 Jump and Branch Instructions 4-24
4.3.5 Trap Instructions 4-28
4.3.6 Special Instructions 4-29
4.3.7 Coprocessor Instructions 4-29
4.3.8 System Control Coprocessor (CP0) Instructions 4-30
4.3.9 Cache Maintenance Instructions 4-31
4.3.10 APU and CW4011 Instruction Set Extensions 4-31
4.3.11 ATMizer II+ Instruction Set Extensions 4-42
4.4 CP0 Data Manipulation Registers 4-49
4.4.1 Rotate Register (23) 4-49
4.4.2 Circular Mask Register (24) 4-49
4.5 Cache Memory 4-50
4.5.1 Cache States 4-50
4.5.2 Address and Cache Tags 4-52
4.5.3 D-Cache Scratch-Pad RAM Mode 4-53
4.5.4 I-Cache RAM Mode 4-54
4.5.5 Cache Instructions 4-55
4.6 Exceptions 4-59
4.6.1 R3000 Exception Compatibility Mode 4-61
4.6.2 Exception Handling Registers 4-62
4.6.3 CW4011 Exceptions 4-80
4.7 Memory Map 4-95
4.7.1 Operating Modes 4-95
4.7.2 ATMizer II+ Chip Memory Map 4-96
4.7.3 Hardware Registers Map 4-98

4.7.4 ATMizer II+ Chip Primary and Secondary Port Access 4-98
4.8 Interrupts 4-101
4.8.1 External Nonvectored Interrupts 4-101
4.8.2 External Vectored Interrupt Sources 4-103
4.8.3 Enabling Vectored Interrupts 4-106
4.8.4 Vectored Interrupt Processing 4-107
4.8.5 Status Checking 4-108
4.8.6 Coprocessor Condition Signals 4-112
4.9 CW4011 OCA Bus Accesses 4-112
4.10 Bus Watchdog Timers 4-112
4.10.1 SC Bus Watchdog Timer 4-113
4.10.2 OCA Bus Watchdog Timer 4-113
4.10.3 APU Priority Register 4-114
4.10.4 APU_Error Register 4-115
4.10.5 OCA Error Register 4-117
4.11 Boot Procedures 4-118
4.11.1 Boot Location 4-118
4.11.2 Serial Interface Boot Sequence 4-119
4.11.3 Cell Buffer Memory Boot Sequence 4-119
4.11.4 Secondary EPROM Boot Sequence 4-119
4.11.5 APU Access to Serial EPROM 4-119

Chapter 5 Enhanced DMA
5.1 Overview 5-1
5.2 Data Structures 5-5
5.2.1 VC Descriptor Structure 5-6
5.2.2 Buffer Descriptor 5-16
5.3 EDMA Commands 5-21
5.3.1 RxCell Command 5-23
5.3.2 TxCell Command 5-24
5.3.3 Buff Command 5-25
5.3.4 Move Command 5-28
5.3.5 TxConClose/RxConClose Command 5-32
5.3.6 Checking Status 5-32
5.3.7 Buffer Completion 5-35
5.4 Data Structure Locations 5-43

5.4.1 VC Descriptors Address Calculation 5-44
5.4.2 Buffer Descriptors 5-46
5.4.3 Buffer Payload 5-48
5.5 Register Descriptions 5-49
5.5.1 EDMA Control Register 5-52
5.5.2 EDMA Error Mask Register 5-53
5.5.3 EDMA Bus Error Register 5-56
5.6 AAL5 Mode Operation 5-57
5.6.1 Transmit Cell Processing Requests 5-57
5.6.2 Receive Cell Processing Requests 5-59
5.6.3 Free Buffers 5-60
5.6.4 Big Endian and Little Endian 5-61
5.7 AAL0 Mode Operation 5-62

Chapter 6 ATM Cell Interface

6.1 ACI Overview 6-2
6.2 Cell Size and Layout 6-4
6.3 Cell Descriptor 6-5
6.4 Memory-Mapped ACI Registers 6-9
6.4.1 ACI_Ctrl Register 6-10
6.4.2 ACI_FreeList Register 6-12
6.4.3 ACI_TxTimer Register 6-13
6.4.4 ACI_TxSize Register 6-14
6.4.5 ACI_TxLimit and ACI_RxLimit Registers 6-14
6.4.6 ACI_RxMask Register 6-14
6.4.7 ACI_Free Register 6-14
6.4.8 ACI_RxRead Register 6-15
6.4.9 ACI_TxWrite Register 6-15
6.4.10 ACI_RxCells and ACI_TxCells Registers 6-16
6.4.11 ACI_Error Register 6-16
6.4.12 ACI_RxSize Register 6-17
6.4.13 ACI_BadHEC Register 6-17
6.4.14 ACI_ClearBytes Register 6-18
6.4.15 ACI_FreeCount Register 6-18
6.5 Cell Buffer Manager 6-18
6.5.1 Cell Buffer Initialization 6-19
6.5.2 Requesting and Releasing a Free Cell Location 6-19

6.5.3 Inserting and Removing Cells from the ACI FIFO 6-20
6.5.4 Setting and Checking FIFO Sizes 6-21
6.6 ACI Receiver 6-21
6.6.1 ACI Receiver Operations 6-22
6.6.2 Receive FIFO Status 6-23
6.6.3 Receive Priority Scheme 6-23
6.6.4 HEC Processing 6-24
6.6.5 CRC10 Verifications 6-24
6.6.6 Utopia Parity Checking 6-25
6.7 ACI Transmitter 6-25
6.7.1 ACI Transmitter Operations 6-25
6.7.2 Transmit FIFO Status 6-26
6.7.3 Idle Cell Generation 6-26
6.7.4 PHY Port Selection and Port Polling 6-27
6.7.5 HEC Generation 6-27
6.7.6 CRC10 Generation 6-28
6.7.7 ACI Transmitter Time-Out 6-28
6.7.8 Utopia Parity Generation 6-28
6.8 Polling Scheme 6-29
6.9 Loopback Mode 6-29
6.10 Utopia Interface 6-30
6.10.1 Utopia Clocks 6-30
6.10.2 Unused Pins 6-30

Chapter 7 Scheduler Unit
7.1 Scheduler Overview 7-1
7.2 Priority Mode Operation 7-3
7.2.1 Example of Priority Mode Operation 7-3
7.2.2 Service Command 7-5
7.2.3 Schedule Command 7-6
7.2.4 Tic Command 7-8
7.3 Flat Mode Operation 7-8
7.3.1 Example of Flat Mode Operation 7-8
7.3.2 Service Command 7-9
7.3.3 Schedule Command 7-10
7.3.4 Tic Command 7-11
7.4 Calendar Switching 7-11

7.5 Command Execution 7-13
7.6 Register Descriptions 7-14
7.6.1 Scheduler Control Register 7-15
7.6.2 Calendar Size Register 7-15
7.6.3 SCD_Now Register 7-15
7.6.4 SCD_Serv, SCD_Sched, and SCD_Tic Registers 7-16
7.6.5 SCD_HeadSel Register 7-16
7.6.6 SCD_Err Register 7-16
7.6.7 SCD_Class0–5 Registers 7-17
7.6.8 Calculating a VC Descriptor Address 7-18
7.6.9 Calculating a Calendar Table Address 7-19

Chapter 8 Timer Unit
8.1 Introduction 8-1
8.2 Timer Clock Selection 8-3
8.2.1 TM_ClockSel Register 8-4
8.2.2 TM_ClockSel2 Register 8-4
8.3 Time-Out Events 8-5

Chapter 9 PCI Interface
9.1 PCI Interface Overview 9-1
9.2 PCI Configuration Space Registers 9-4
9.2.1 Vendor ID Register 9-6
9.2.2 Device ID Register 9-6
9.2.3 Command Register 9-7
9.2.4 Status Register 9-8
9.2.5 Revision ID Register 9-10
9.2.6 Class Code Register 9-10
9.2.7 Cache Line Size Register 9-11
9.2.8 Latency Timer Register 9-11
9.2.9 Header Type Register 9-12
9.2.10 Base Address Register 1 9-12
9.2.11 Base Address Register 2 9-13
9.2.12 Subsystem Vendor ID Register 9-15
9.2.13 Subsystem ID Register 9-15
9.2.14 Interrupt Line Register 9-16

9.2.15 Interrupt Pin Register 9-16
9.2.16 Minimum Grant Register 9-17
9.2.17 Maximum Latency Register 9-17
9.2.18 TRDY_Timer Register 9-18
9.2.19 Retry_Timer Register 9-18
9.2.20 Configuration Target Operation 9-19
9.2.21 Configuration Master Operation 9-20
9.3 Primary Port Registers 9-21
9.3.1 XPP_Ctrl Register 9-22
9.3.2 PP_Ctrl Register 9-24
9.3.3 Primary Port Slave Prefetch Register 9-25
9.3.4 Primary Port Error Register 9-27
9.3.5 Primary Port Error Address Register 9-29
9.4 PCI Slave Transactions 9-29
9.4.1 Mailbox 9-30
9.4.2 PCI Slave Write Timing 9-32
9.4.3 PCI Slave Read Timing 9-34
9.4.4 PCI Slave Errors 9-36
9.5 PCI Master Transactions 9-36
9.5.1 PCI Master Write Timing 9-37
9.5.2 Master Write Errors 9-39
9.5.3 PCI Master Read Timing 9-40
9.5.4 Master Read Errors 9-43
9.6 Balancing Bus Usage 9-44
9.6.1 Master Write 9-44
9.6.2 Master Read 9-45
9.6.3 Slave Write 9-45
9.6.4 Slave Read 9-46

Chapter 10 Secondary Bus Memory Controller
10.1 Overview 10-2
10.2 SBC Configuration 10-3
10.2.1 SP_Ctrl Register 10-4
10.2.2 Secondary Bus Clock Control Register 10-6
10.3 Secondary Bus Performance Considerations 10-11
10.4 SDRAM Controller 10-14
10.4.1 SDRAM Connections 10-14

10.4.2 SDRAM Controller Configuration 10-16
10.4.3 SDRAM Initialization 10-19
10.4.4 SDRAM Refresh 10-20
10.4.5 Secondary Bus Time-Out 10-22
10.4.6 SDRAM Command Summary 10-23
10.4.7 SDRAM Read Transfer 10-23
10.4.8 SDRAM Write Transfer 10-24
10.5 SSRAM Controller 10-25
10.5.1 SSRAM Read Transfers 10-27
10.5.2 SSRAM Write Transfers 10-27
10.6 32-Bit SRAM/EPROM Controller 10-28
10.6.1 32-Bit SRAM/EPROM Read Transfer 10-28
10.6.2 32-Bit SRAM Write Transfers 10-29
10.6.3 32-Bit SRAM/EPROM SB_RDYn Timing 10-30
10.7 PHY Controller 10-32
10.7.1 PHY Read Transfers 10-32
10.7.2 PHY Write Transfers 10-33
10.7.3 PHY SB_RDYn Timing 10-34
10.8 8-Bit SRAM/EPROM Controller 10-36
10.8.1 8-Bit SRAM/EPROM Read Transfers 10-36
10.8.2 8-Bit SRAM/EPROM Write Transfers 10-37
10.8.3 8-Bit SRAM/EPROM SB_RDYn Timing 10-38
10.9 External Bus Masters 10-40
10.10 Error Reporting 10-41

Chapter 11 System Clock
11.1 System Clock Options 11-1
11.2 Clock Synthesis 11-2
11.3 Design Considerations 11-4

Chapter 12 JTAG Interface
12.1 JTAG Instructions 12-1
12.1.1 BYPASS Instruction 12-2
12.1.2 SAMPLE/PRELOAD Instruction 12-2
12.1.3 EXTEST Instruction 12-2
12.1.4 HI-Z Instruction 12-3
12.2 Boundary Scan Chain Order 12-3

Chapter 13 **Specifications**
13.1 AC Timing 13-1
13.2 Electrical Requirements 13-10
13.2.1 I/O Pad Drivers and Receivers 13-10
13.2.2 I/O Level Requirements 13-13
13.3 Pin Summary 13-14
13.4 Package Information 13-15

Appendix A **Register Summary**

Appendix B **The ATM Cell**
B.1 ATM Cell Structure B-1
B.2 The AAL5 Trailer B-3

Appendix C **Glossary of Abbreviations**

Customer Feedback

Figures
2.1 L64364 Functional Block Diagram 2-2
3.1 I/O Signals (Utopia Master) 3-2
3.2 I/O Signals (Utopia Slave) 3-3
4.1 APU Block Diagram 4-2
4.2 CW4011 Block Diagram 4-5
4.3 CW4011 Instruction Pipeline 4-7
4.4 Instruction Formats 4-14
4.5 Byte Specifications for Loads/Stores 4-15
4.6 Rotate Register 4-49
4.7 CMask Register 4-49
4.8 I-Cache and D-Cache State Diagram 4-51
4.9 D-Cache Write Back State Diagram 4-52
4.10 Cache Address Format 4-53
4.11 Tag RAM Access Format 4-54
4.12 Cache Instruction Format 4-55
4.13 Tag Test Mode Format 4-59

4.14	DCS Register	4-63
4.15	Count Register	4-64
4.16	Compare Register	4-65
4.17	Status Register (R4000 Mode)	4-65
4.18	Status Register (R3000 Mode)	4-68
4.19	Cause Register	4-71
4.20	EPC Register	4-73
4.21	PRId Register	4-73
4.22	CCC Register	4-74
4.23	LLAdr Register	4-77
4.24	BPC Register	4-78
4.25	BDA Register	4-78
4.26	BPCM Register	4-78
4.27	BDAM Register	4-79
4.28	Error EPC Register	4-79
4.29	Cold Reset Exception	4-81
4.30	Warm Reset, NMI Exceptions	4-81
4.31	Common Exceptions	4-82
4.32	Debug Exception	4-82
4.33	External Vectored Interrupt Exception	4-83
4.34	CW4011 Virtual Memory Map	4-96
4.35	APU_AddrMap Register	4-99
4.36	Primary Port Address Formation	4-100
4.37	Secondary Address Formation for Exception Vectors	4-101
4.38	APU_VIntEnable Register	4-106
4.39	APU_VIntBase Register Format	4-107
4.40	APU Status Register Format	4-108
4.41	APU_SCbus_Watchdog Register	4-113
4.42	APU_OCAbus_Watchdog Register	4-114
4.43	APU_Priority Register	4-115
4.44	APU_Error Register	4-115
4.45	OCA_Err Register	4-117
5.1	EDMA Processors	5-2
5.2	Virtual Connection and Buffer Descriptors	5-5
5.3	Virtual Connection Descriptor	5-7
5.4	VC Descriptor Control Field	5-11
5.5	BuffPres and ConAct Bits Timing	5-13
5.6	VCD_RxCtrl Usage	5-14

5.7	Buffer Descriptor	5-16
5.8	Buffer Descriptor Control Field	5-19
5.9	EDMA Request & Completion Queues	5-21
5.10	EDMA_RxCell Register Format	5-23
5.11	EDMA_TxCell Register Format	5-25
5.12	EDMA_Buff Register Format	5-26
5.13	EDMA_MoveSrc and EDMA_MoveDst Register Format	5-28
5.14	EDMA_MoveCount Register	5-29
5.15	EDMA_MoveCount2 Register	5-31
5.16	Tx/RxConClose Command Format	5-32
5.17	EDMA_Status Register	5-33
5.18	Primary Completion Queue	5-36
5.19	Auxiliary Completion Queue	5-36
5.20	Buffer Status Bits	5-37
5.21	TX/RX_EDMA_VCD_Base Register	5-44
5.22	VC Descriptor Address Calculation for PCI Memory	5-45
5.23	VC Descriptor Address Calculation for Local or Cell Buffer Memory	5-46
5.24	Buffer Descriptor Address Calculation	5-46
5.25	EDMA_BFD_FBase Register	5-47
5.26	EDMA_BFD_LBase Register	5-47
5.27	EDMA_Ctrl Register	5-52
5.28	EDMA_ErrMask Register	5-54
5.29	EDMA_BusErr Register	5-56
5.30	Byte Swapping	5-62
5.31	VC Descriptor Control Fields (AAL0 Mode Uses CRC32 Field)	5-63
6.1	ACI Block Diagram	6-3
6.2	Cell Layout	6-5
6.3	Cell Descriptor Format	6-5
6.4	ACI_Ctrl Register	6-10
6.5	ACI_Free List Register	6-13
6.6	ACI_TxTimer Register Format	6-13
6.7	ACI_TxWrite Register	6-16
6.8	ACI_BadHEC Register	6-17
7.1	VC Descriptor Format (Word 0)	7-3
7.2	Scheduler Calendar Table in Priority Mode	7-3
7.3	Priority Mode - Calendar Table	7-4

7.4 Priority Mode - Calendar Table 7-5
7.5 Service Command Return Value 7-6
7.6 SCD_Sched Register Format 7-6
7.7 Format of SCD_HeadSel Register 7-7
7.8 Priority Mode - Calendar Table 7-7
7.9 Flat Mode - Calendar Table after Schedule Command 7-8
7.10 Flat Mode - Calendar Table after Tic Command 7-9
7.11 Flat Mode - Calendar Table with SCD_HeadSel0 Bit Set 7-10
7.12 Format of the SCD_CalSwitch Register 7-11
7.13 Flat Mode - Calendar Table 0 7-12
7.14 Flat Mode - Calendar Table 1 7-12
7.15 Scheduler Control Register Format 7-15
7.16 SCD_Err Register 7-17
7.17 SCD_Class0–5 Registers Format 7-17
7.18 VC Descriptor Address Computations 7-18
7.19 Calendar Table Address Computations 7-19
8.1 Timer Clock Selection Registers Format 8-3
9.1 PCI Interface Block Diagram 9-3
9.2 PCI Configuration Space Registers 9-5
9.3 Vendor ID Register 9-6
9.4 Device ID Register 9-6
9.5 Command Register 9-7
9.6 Status Register 9-8
9.7 Revision ID Register 9-10
9.8 Class Code Register 9-10
9.9 Cache Line Size Register 9-11
9.10 Latency Timer Register 9-11
9.11 Header Type Register 9-12
9.12 Base Address Register 1 9-12
9.13 Base Address Register 2 9-13
9.14 Subsystem Vendor ID Register 9-15
9.15 Subsystem ID Register 9-15
9.16 Interrupt Line Register 9-16
9.17 Interrupt Pin Register 9-16
9.18 Minimum Grant Register 9-17
9.19 Maximum Latency Register 9-17
9.20 TRDY_Timer Register 9-18
9.21 Retry_Timer Register 9-18

9.22 Configuration Space Read 9-20
9.23 Configuration Space Write 9-20
9.24 XPP_Ctrl Register 9-22
9.25 PP_Ctrl Register 9-24
9.26 PP_SlavePFtch Register 9-26
9.27 PP_Err Register 9-27
9.28 PP_ErrAddr Register 9-29
9.29 Mailbox Registers 9-31
9.30 Slave Write Timing 9-33
9.31 Slave Write Stop Timing 9-33
9.32 Parity Error Timing 9-34
9.33 PCI Slave Read Timing 9-35
9.34 Master Write Timing 9-37
9.35 Master Write Stop Timing 9-38
9.36 PCI Master Read Timing 9-41
9.37 Master Read Stop Timing 9-42
9.38 Master Read Error Timing 9-42
10.1 SP_Ctrl Register 10-4
10.2 SB Clock Relationships 10-7
10.3 Secondary Bus Clock Control Register 10-7
10.4 Effects of SB_DCLK Delay Register 10-9
10.5 Effects of SB_CLKO Delay Register 10-11
10.6 SP_SDRAM Register 10-16
10.7 SDRAM Mode Register 10-19
10.8 SP_Refresh Register 10-20
10.9 SDRAM Refresh Timing 10-22
10.10 SDRAM Read Timing 10-24
10.11 SDRAM Write Timing 10-25
10.12 SSRAM Read Timing 10-27
10.13 SSRAM Write Timing 10-28
10.14 SRAM Read Timing 10-29
10.15 SRAM Write Timing 10-30
10.16 32-Bit SRAM/EPROM Read Timing with SB_RDYn 10-31
10.17 32-Bit SRAM/EPROM Write Timing with SB_RDYn 10-31
10.18 PHY Read Timing 10-33
10.19 PHY Write Timing 10-34
10.20 PHY Read Timing with SB_RDYn 10-35
10.21 PHY Write Timing with SB_RDYn 10-35

10.22 8-Bit SRAM/EPROM Read Timing 10-37
10.23 8-Bit SRAM Write Timing 10-38
10.24 8-Bit SRAM/EPROM Read Timing with SB_RDYn 10-39
10.25 8-Bit SRAM/EPROM Write Timing with SB_RDYn 10-39
10.26 Secondary Bus Grant Timing 10-40
10.27 SB_Err Register 10-41
10.28 SB_ErrAddr Register 10-42
11.1 Clock Selection and Synthesis Circuit 11-2
11.2 Phase-Locked Loop 11-3
11.3 PLL Supply Filtering 11-4
13.1 Output Signal Timing Reference Points 13-1
13.2 Input Signal Timing Reference Points 13-2
13.4 240-pin PQUAD (NL) Mechanical Drawing 13-17
B.1 The ATM Cell Layout at the UNI B-1
B.2 The AAL5 Trailer Layout B-3

Tables

2.1 Big/Little Endian Mapping 2-6
4.1 APU Instruction Set Summary 4-10
4.2 Load and Store Instructions Summary 4-16
4.3 Load and Store Instruction Summary–MIPS II ISA Extensions 4-17
4.4 ALU Immediate Instruction Summary 4-18
4.5 Three-Operand, Register Type-Instruction Summary 4-19
4.6 Shift Instruction Summary 4-20
4.7 Multiply/Divide Instruction Summary 4-21
4.8 Execution Time of Multiply and Divide Instructions 4-21
4.9 Instruction Set Extensions 4-22
4.10 CW4011 ISA Extensions Summary 4-23
4.11 APU Rate Instruction Extensions 4-24
4.12 Jump Instruction Summary 4-25
4.13 Branch Instruction Summary 4-26
4.14 Branch-Likely Instruction Summary–MIPS II ISA Extensions 4-27
4.15 Trap Instruction Summary–MIPS II ISA Extensions 4-28
4.16 Special Instruction Summary 4-29
4.17 Coprocessor Instruction Summary 4-29
4.18 CP0 Instruction Summary 4-30

4.19	CP0 Instruction Extension Summary	4-30
4.20	Cache Maintenance Instruction Summary	4-31
4.21	D-Cache Write-Back Mode	4-51
4.22	Cache Control Bits	4-56
4.23	TAG and INV Encoding	4-58
4.24	APU Exceptions	4-60
4.25	CP0 Exception Processing Registers	4-62
4.26	Exception Codes	4-72
4.27	Exception Vector Base Addresses	4-84
4.28	Exception Vector Offset Addresses	4-84
4.29	Exception Priority Order	4-85
4.30	Segment Properties	4-96
4.31	ATMizer II+ Chip Memory Map	4-97
4.32	ATMizer II+ Chip Hardware Register Map	4-98
4.33	Nonvectored Interrupt Sources	4-102
4.34	Vectored Interrupt Sources	4-104
4.35	Coprocessor Condition Signals	4-112
4.36	Boot Sequence	4-118
5.1	EDMA Commands	5-3
5.2	VC Descriptor Fields	5-8
5.3	VC Descriptor Control Bits	5-12
5.4	Buffer Descriptor Fields	5-16
5.5	Buffer Descriptor Control Bits	5-19
5.6	BFS_BuffFree, BFS_BuffLarge, and BFS_FreeSel Encoding	5-27
5.7	Tx Completion Queue Messages	5-42
5.8	Rx Completion Queue Messages	5-42
5.9	Buff Completion Queue Messages	5-43
5.10	EDMA Memory Mapped Registers	5-49
6.1	Cell Size	6-4
6.2	Memory Mapped ACI Registers	6-9
7.1	Scheduler Registers	7-14
8.1	Timer Unit Registers	8-2
9.1	PCI FIFO's	9-4
9.2	ATMizer II+ Chip External Memory Map	9-30
10.1	16 Mbyte Secondary Bus Memory Map	10-3
10.2	64 Mbyte Secondary Bus Memory Map	10-3
10.3	SBC Clocks per Data Word	10-13

10.4 SBC Transfer Lead-Off Cycles 10-13
10.5 ATMizer II+ Chip to SDRAM Interconnections 10-15
10.6 SDRAM Command Summary 10-23
10.7 SSRAM Configurations 10-26
10.8 SSRAM Interconnections 10-26
10.9 Secondary Bus to PHY Device Connections 10-32
11.1 Loop Filter Components 11-3
12.1 JTAG Instruction Register Encoding 12-2
12.2 L64364 Boundary Scan Chain 12-3
13.1 PCI Interface Timing 13-2
13.2 Secondary Bus Timing 13-5
13.3 Utopia Interface Transmit Timing 13-6
13.4 Utopia Interface Receive Timing 13-8
13.5 Miscellaneous Timing 13-9
13.6 I/O Pad Drivers and Receivers 13-10
13.7 DC Characteristics 13-13
13.8 L64364 Pin Summary 240 Pin Alphabetical Pin List 13-15
13.9 PQUAD Electrical and Thermal Data 13-17

Chapter 1
Introduction

This chapter introduces the LSI Logic L64364 Application Specific Standard Product (ASSP) and includes the following sections:

- Section 1.1, “Overview,” page 1-1
- Section 1.2, “Functional Description,” page 1-2
- Section 1.3, “Features,” page 1-3

1.1 Overview

The L64364 Application-Specific Standard Product is a highly integrated ATM Segmentation and Reassembly (SAR) engine optimized for inter-networking applications. PX80 is an 80 MHz speed grade ATMizer II+ and PX100 is a 100 MHz speed grade ATMizer II+. The L64364 is the third generation device within the LSI Logic ATMizer product family and offers important enhancements over the previous L64363 chip.

A primary feature of the ATMizer product family is the flexibility offered by the ATM Processing Unit (APU), which is based on an embedded MIPS processor. The APU allows you to modify device behavior by downloading new code to accommodate changes in ATM standards, particularly enhancements and adaptations of flow control algorithms. A specific set of hardware functional blocks, including a complete SAR controller, support the APU. These blocks off-load repetitive data manipulation tasks from the APU, allowing it to achieve full-duplex, 155 Mbits/s performance levels.

The L64364 is currently implemented in the LSI Logic G10®-p 0.35-micron CMOS process and is available in a 240-pin PQUAD package.

1.2 Functional Description

The L64364 is designed to provide 155 Mbits/s of full-duplex operation while performing segmentation and reassembly of ATM Adaptation Layer 5 (AAL5) CS-PDUs. A specialized hardwired AAL5 SAR engine, called the Enhanced DMA (EDMA), assists the MIPS-based ATM Processing Unit in segmentation and reassembly tasks and memory management functions.

Although the EDMA is responsible for all basic segmentation and reassembly functions, it operates under full control of the APU. The APU is responsible for traffic management, host messaging, and any other upper layer tasks. As an option, the advanced functions of the hardwired units may be switched-off to give the APU full control of all operations. However, this will impact overall performance.

The ATM Processing Unit is based upon the LSI Logic MIPS II compatible CW4011 MiniRISC® microprocessor core. The processor delivers 160 MIPS peak (110 MIPS sustained) when operating at 80 MHz and 200 MIPS peak (138 MIPS sustained) when operating at 100 MHz. The APU instruction set is extended with ATM specific instructions to enhance performance. These instructions accelerate the cell rate calculations for Available Bit Rate (ABR) services by allowing direct arithmetic operations (add, subtract, and multiply) on rates expressed as ATM Forum floating point 15-bit numbers.

Scheduling and policing of different ATM Quality of Service (QoS) connections can be achieved efficiently with the help of the integrated hardware Scheduler that supports six priority classes. The Scheduler uses calendar tables to create arbitrary traffic schemes to a limit of 64 K Virtual Connections (VCs).

The primary interface for the device is a 33 MHz, 32-bit wide, PCI Bus. As the bus master, the L64364 is able to autonomously access control and data structures located in the system memory. As a bus slave, the device provides transparent access to local memory, internal Cell Buffer Memory (CBM), and internal hardware registers for external PCI Bus masters. The PCI interface implements four separate FIFOs to maximize the performance of simultaneous read/write operations as bus master or slave.

The L64364 integrates a memory controller that provides a glueless interface for asynchronous SRAMs, synchronous SRAMs, and synchronous DRAMs that are used for local memory; it can also serve as an interface to external physical layer devices such as framers. The memory controller allows APU booting from parallel, byte-wide EPROMs and from serial EPROMs.

The device includes a JTAG controller and boundary scan logic to simplify board-level tests.

1.3 Features

Key features of the L64364 are:

- Supports full-duplex, OC-3 (155 Mbits/s) rate.
- Processes AAL1, AAL3/4, and AAL5 protocol layers.
- Includes scatter/gather EDMA, which supports fragmented and unaligned host data buffers.
- 64 K Virtual Connections.
- Supports operation and management functions.
- Supports a maximum of six traffic types.
- Supports all flow control algorithms, including the ATM Forum rate-based scheme.
- Provides flexible ATM cell header translation mechanism.
- Performs error monitoring, statistics gathering, host messaging, and diagnostics.
- 33 MHz, 32-bit, PCI 2.1 compliant interface bus.
- 50 MHz, 8-bit, Utopia Level 2 compliant interface bus.

Chapter 2
Functional Overview

This chapter provides an overview of the L64364 ATMizer II+ chip functional units. Separate sections summarize the features of each functional unit. Subsequent chapters in this manual provide more detailed descriptions of each unit.

This chapter contains the following sections:

- Section 2.1, “Major Functional Units,” page 2-1
- Section 2.2, “ATM Processing Unit (APU),” page 2-3
- Section 2.3, “Enhanced DMA (EDMA),” page 2-3
- Section 2.4, “ATM Cell Interface (ACI),” page 2-4
- Section 2.5, “Scheduler and Timer Units,” page 2-5
- Section 2.6, “PCI Interface,” page 2-5
- Section 2.7, “Endian Considerations,” page 2-6
- Section 2.8, “Secondary Bus Memory Controller (SBC),” page 2-7
- Section 2.9, “Other Features,” page 2-7

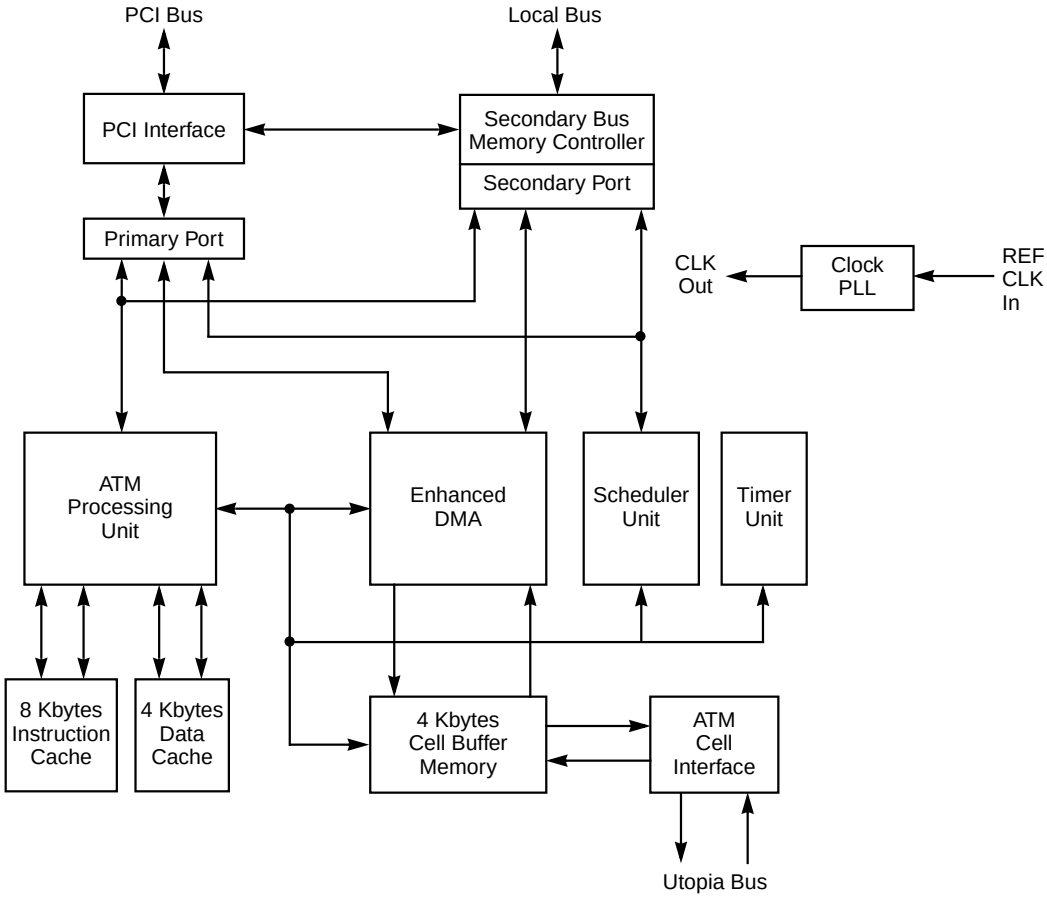
2.1 Major Functional Units

As shown in Figure 2.1, the L64364 includes the following major functional units:

- Enhanced DMA (EDMA)
- ATM Processing Unit (APU)
- Scheduler Unit
- Timer Unit
- ATM Cell Interface (ACI)

- Primary Port Interface
- PCI Interface
- Secondary Port and Bus Memory Controller (SBC)
- Cell Buffer Memory (CBM)
- Separate APU Instruction and Data Caches/RAM
- JTAG Controller
- Clock Phase-Locked Loop (PLL)

Figure 2.1 L64364 Functional Block Diagram



2.2 ATM Processing Unit (APU)

The APU is based on the CW4011 MiniRISC Core. APU features include the following:

- MIPS II MiniRISC Processor with ATM specific extensions.
- 160 native MIPS peak, 110 native MIPS sustained at 80 MHz and 200 native MIPS peak, 138 native MIPS sustained at 100 MHz.
- Custom floating point processor for ATM rate calculations.
- 8 Kbytes Instruction Cache implemented in two sets. Each set configurable as cache or RAM.
- 4 Kbytes Data Cache implemented in two sets. Each set configurable as cache or RAM.
- APU boot options include external serial EPROM, external byte-wide EPROM, or host download to the ATMizer's Cell Buffer Memory.
- Messaging mailbox for host-to-ATMizer APU communications.

2.3 Enhanced DMA (EDMA)

The EDMA is a complete ATM Adaptation Layer 5 Segmentation and Reassembly (AAL5 SAR) engine with hardware assists for other AALs. The EDMA autonomously performs SAR tasks, freeing the APU for scheduling and user value add functions. EDMA features include the following.

- AAL5 SAR engine capable of sustained 155 Mbytes/s cell throughput.
- Hardware support for AAL1 and AAL3/4 SAR functions.
- Scatter/gather DMA support for fragmented or misaligned host data buffers.
- Support for 64 K Virtual Connections (practical limit set by external memory).
- Support for 64 K payload buffers (practical limit set by external memory).

- Independent Transmit, Receive, Buffer, and Move engines optimize memory bandwidth utilization.
- Optional byte swapping to support little endian hosts.
- Programmable cell length of 53–65 bytes.
- Optional cell header insertion/deletion.
- CRC-32 generation/checking for AAL5.
- UU/CPI support for AAL5.
- DMA Move command supports packet transfers from PCI to Secondary Memory.
- Insertion and extraction of congestion notification, EFCI and CLP, on a per-cell or per-CS-PDU basis.

2.4 ATM Cell Interface (ACI)

The ACI consists of a Utopia, Multi-PHY, eight-bit interface, and the Cell Buffer Manager. The Cell Buffer Manager maintains a transmit FIFO, receive FIFO, and error FIFO within the 4 Kbyte Cell Buffer Memory (CBM). ACI features include:

- *ATM Forum Utopia Level 2, Version 1.0* compliant interface configurable as either master or slave.
- Supports cell-level handshaking per *ATM Forum Utopia Level 2, Version 1.0*.
- 50 MHz maximum Utopia `Tx_Clk`, `Rx_Clk` rate.
- Multi-PHY polling configurable as round-robin or fixed priority.
- Multi-PHY polling configurable as direct (4 slave devices maximum) or multiplexed (24 slave devices maximum).
- Cell FIFO size configurable up to 4 Kbytes.
- Optional HEC generation and checking.
- CRC-10 generation and checking for AAL3/4.
- Optional idle cell insertion.
- Cell loopback.
- Optional Utopia parity generation and checking.

- Utopia transmit timer to prevent blocking by unresponsive slave in Multi-PHY applications.

2.5 Scheduler and Timer Units

The Scheduler block provides hardware support for traffic scheduling algorithms. The Timer block consists of one 32-bit Time Stamp Counter and 8 general-purpose timers used to support other traffic shaping or maintenance functions. Specific features include:

- Traffic scheduling on a per-VC basis (64 K VCs maximum).
- Support for up to six priorities in traffic scheduling queues.
- Scheduler maintains Scheduler Calendar table in external secondary memory or internal CBM.
- Eight, 8-bit, cascadable, general-purpose timers with optional clock source including external clock.
- A 32-bit Time Stamp Counter with optional clock source including external clock.

2.6 PCI Interface

The PCI Interface is designed for write-optimized systems architected for either cell or packet transfers over the PCI Bus. A high performance bridge allows the PCI host to burst transmit packets to secondary memory (slave write). The EDMA bursts receive packets to the PCI after reassembly in secondary memory (master write). Master read and slave read transfers are also supported. PCI Interface features include:

- PCI Protocol conforms to *PCI Local Bus Specification, Revision 2.1*.
- 33 MHz, 32-bit PCI Bus support.
- Four independent PCI FIFOs—Master Read, Master Write and Slave Write FIFOs are each 128 bytes; Slave Read FIFO is 32 bytes.
- ATMizer II+ chip's CBM, Messaging Mailbox, and internal hardware registers mapped to PCI memory space.
- Secondary Memory (16 or 64 Mbytes) mapped to PCI memory space.

2.7 Endian Considerations

The ATMizer II+ chip operates internally in big endian mode. The ATMizer II+ chip's external buses, however, are little endian format. Internal master requests to the external PCI Bus and local Secondary Bus (SB) are mapped such that the internal masters see the buses as big endian resources. Similarly, external PCI host (little endian) requests to the ATMizer II+ chip's internal and SB resources are mapped such that the PCI host sees little endian resources.

This is done to facilitate the transfer of word-based control structures between big and little endian domains. In both endians, word addresses point to the same 32-bit word (data[31:0]). This allows big and little endian masters to manipulate control structures with respect to their own domains. There is no need for byte swapping the control structures or translating pointers to control structures.

Table 2.1 shows how internal big endian and external little endian requests map to the ATMizer II+ chip's three available memory resources: Cell Buffer Memory, PCI Bus, and Secondary Bus.

Table 2.1 Big/Little Endian Mapping

Internal Request (Big Endian)	External PCI Host (Little Endian)	CBM (Big Endian)	PCI (Little Endian)	SB (Little Endian)
data[31:24], be[0]	data[31:24], be[3]	data[31:24], be[0]	data[31:24], be[3]	data[31:24], be[3]
data[23:16], be[1]	data[23:16], be[2]	data[23:16], be[1]	data[23:16], be[2]	data[23:16], be[2]
data[15:8], be[2]	data[15:8], be[1]	data[15:8], be[2]	data[15:8], be[1]	data[15:8], be[1]
data[7:0], be[3]	data[7:0], be[0]	data[7:0], be[3]	data[7:0], be[0]	data[7:0], be[0]

To facilitate the transfer of byte-based buffer payload data between big and little endian masters, the EDMA can swap data-byte positions within a data word to preserve the data byte's address in both endian domains. This applies to data transferred using the EDMA `RxCell`, `TxCell`, and `Move` commands. For the `RxCell` and `TxCell` commands, it is controlled by the `EDMA_ByteSwap` bit in the EDMA_Ctrl register (page 5-53). For the `Move` command, it is controlled by the LEndian bit EDMA_MoveCount2 register (page 5-31). See Section 5.6.4, "Big Endian and Little Endian," for more detail on EDMA byte swapping.

2.8 Secondary Bus Memory Controller (SBC)

The SBC provides access to external memory used for APU instructions and SAR/scheduling data structures. The SBC is also used to access byte-wide EPROM for APU boot and as the management interface for external PHY devices. SBC features include:

- 64 Mbytes maximum memory space.
- Memory controllers for asynchronous SRAM, synchronous SRAM, synchronous DRAM, EPROM, and peripheral devices.
- Arbitration support for external secondary bus master.

2.9 Other Features

The L64364 ATMizer II+ chip also supports the following features:

- IEEE 1149.1 compliant boundary scan.
- Full scan with greater than 99% stuck-at fault coverage in manufacturing test.
- Phase-locked loop to double the frequency of either the 33 MHz PCI clock input or a 40 MHz maximum SYS clock input to use as an internal system clock.

Chapter 3
Signal Descriptions

This chapter describes the primary input and output signals for the L64364 and includes the following sections:

- Section 3.1, “I/O Signals Summary,” page 3-1
- Section 3.2, “PCI Interface,” page 3-4
- Section 3.3, “Secondary Memory Interface,” page 3-8
- Section 3.4, “Utopia Interface,” page 3-10
- Section 3.5, “Clocks and Utility Signals,” page 3-15
- Section 3.6, “APU Signals,” page 3-17
- Section 3.7, “Serial EPROM Interface,” page 3-18
- Section 3.8, “JTAG Test Interface,” page 3-19
- Section 3.9, “Power and Ground Pins,” page 3-20

3.1 I/O Signals Summary

Figure 3.1 illustrates the L64364 input and output signals when using the Utopia master mode. Figure 3.2 illustrates the L64364’s input and output signals when using the Utopia slave mode. The differences are in the Utopia interface and are noted in the signal descriptions following the figures. Those descriptions are grouped into the interfaces shown in the figures.

Signal names for LOW-active signals terminate in lower case “n,” whereas HIGH-active signals and clocks do not. All signals are synchronous to the rising edge of their respective clocks unless specified as asynchronous in the signal descriptions.

Refer to Chapter 13 for PQUAD pinout information.

Figure 3.1 I/O Signals (Utopia Master)

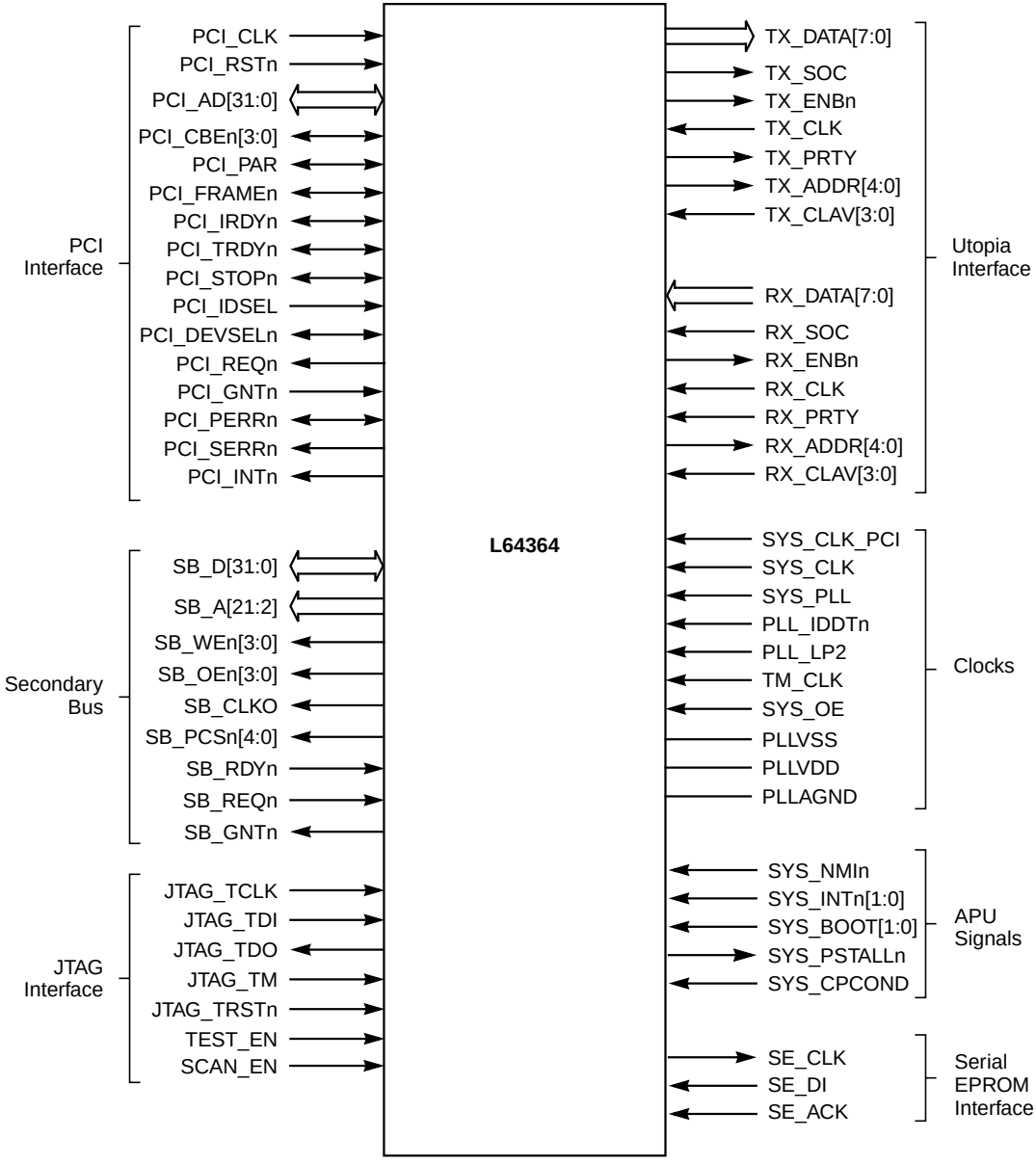
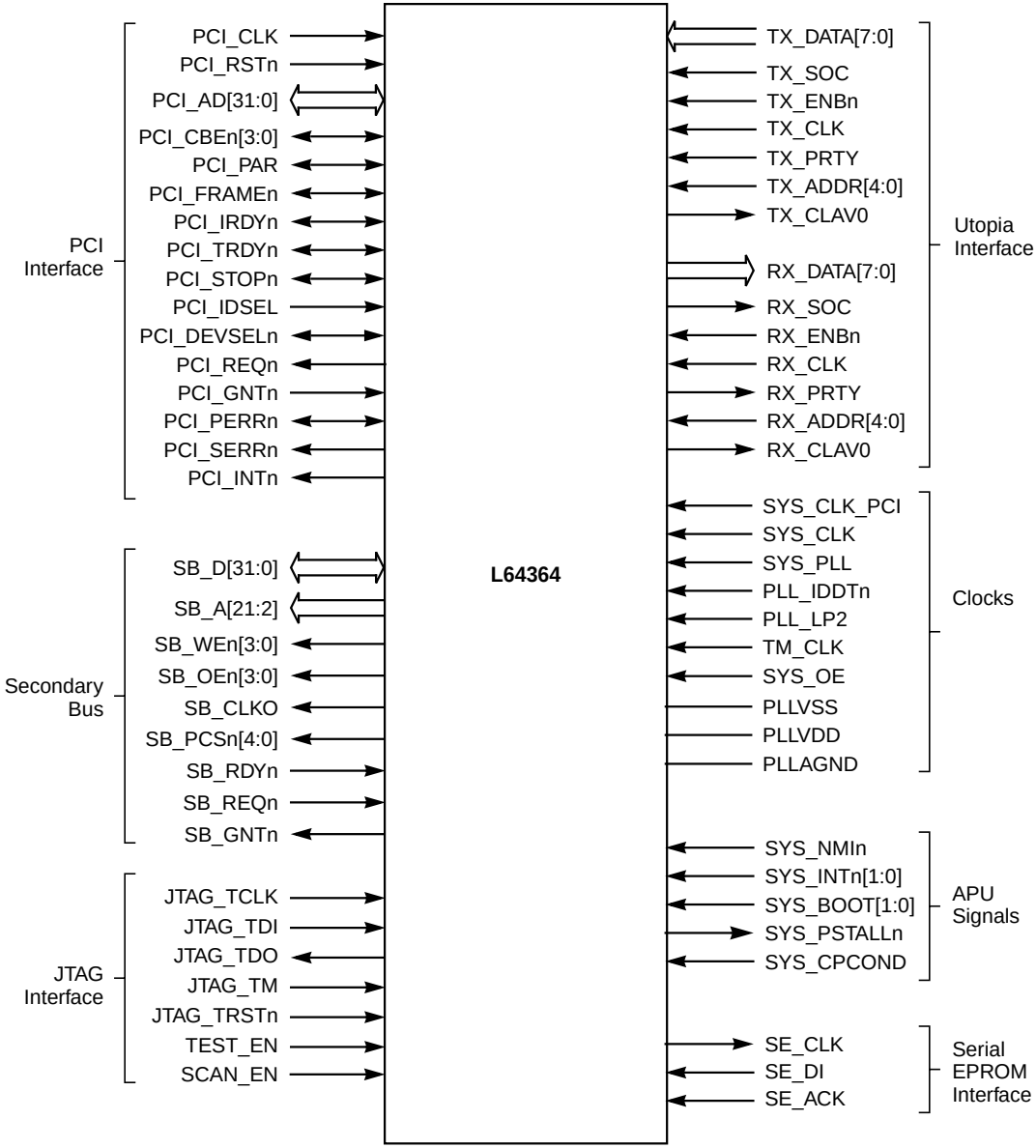


Figure 3.2 I/O Signals (Utopia Slave)



3.2 PCI Interface

The L64364 supports bus master and slave transfers on the PCI Interface. The PCI Interface conforms to *PCI Local Bus Specification 2.1*. Details of PCI Interface functionality are described in Chapter 9.

The PCI Bus is defined as little endian, which means that:

- byte address 0 is the address of the least significant byte (LSB) of the 32-bit PCI data word which is the data byte on PCI_AD[7:0].
- byte address 3 is the address of the most significant byte (MSB) of the 32-bit PCI data word which is the data byte on PCI_AD[31:24].

PCI Interface signal descriptions are provided below.

PCI_CLK **PCI Clock** **Input**

PCI_CLK is a 33 MHz clock that provides timing for all transactions on the PCI Bus. All PCI input signals, except PCI_RSTn, are sampled on the rising edge of PCI_CLK.

PCI_RSTn **PCI Reset** **Input**

PCI_RSTn is an asynchronous signal that resets PCI specific registers, state machines, and signals to an initial state. All PCI Interface signals are held 3-stated when PCI_RSTn is asserted. This signal is also the master reset for the L64364. Deasserting PCI_RSTn causes the APU to initiate the boot process. Refer to Section 4.11, "Boot Procedures," for further information.

PCI_RSTn is asserted to internal control logic asynchronously but deassertion is synchronized to the system clock. When the PLL is used, PCI_CLK or SYS_CLK should be stable for three clock periods before PCI_RSTn is deasserted.

PCI_AD[31:0] **PCI Address and Data Bus** **Bidirectional**

The PCI's 32-bit addresses and 32-bit data are multiplexed on PCI_AD[31:0]. The address phase is the clock cycle in which PCI_FRAMEn is asserted. During the address phase, PCI_AD[31:0] contains a physical memory address.

The LSB of the 32-bit data word is transferred on PCI_AD[7:0], and the MSB is transferred on

PCI_AD[31:24]. Write data is stable when PCI_IRDYn is asserted and read data is stable when PCI_TRDYn is asserted.

PCI_CBEn[3:0] **PCI Bus Commands and Data Byte Enables** **Bidirectional**

The PCI Bus commands and data byte enables are multiplexed on PCI_CBEn[3:0]. During the address phase, PCI_CBEn[3:0] contains the bus commands, some of which are shown in the following table.

PCI_CBEn[3:0]	Command Type
0x6	Memory Read
0x7	Memory Write
0xA	Configuration Read
0xB	Configuration Write
0xC	Memory Read Multiple
0xE	Memory Read Line
0xF	Memory Write and Invalidate

During the data phase, PCI_CBE[3:0] contains the byte enables. PCI_CBE[0] enables data writes on PCI_AD[7:0], the LSB, and PCI_CBE[3] enables data writes on PCI_AD[31:24], the MSB.

PCI_PAR **PCI_Parity** **Bidirectional**

PCI_PAR is set or reset to create even parity for the 36-bits that include PCI_AD[31:0] and PCI_CBEn[3:0]. If the total number of 1's (bits set to logic one) on PCI_AD[31:0] and PCI_CBEn[3:0] is odd, PCI_PAR is set to 1; otherwise, it is reset to 0.

PCI_FRAMEn **PCI Cycle Frame** **Bidirectional**

PCI_FRAMEn is asserted by the current master to indicate the beginning and duration of an access. PCI_FRAMEn is asserted to indicate the start of a bus transaction. Bus transfers continue while PCI_FRAMEn is asserted and they terminate when PCI_FRAMEn is deasserted. When the L64364 deasserts PCI_FRAMEn, it drives it high for one clock cycle and then 3-states it in the following cycles.

PCI_IRDYn **PCI Initiator Ready** **Bidirectional**

The initiator, which is the current bus master, asserts PCI_IRDYn to indicate when there is valid data on PCI_AD[31:0] during a write cycle, or to indicate that it is ready to accept data from PCI_AD[31:0] during a read cycle. A data phase (transfer) is completed on any clock cycle where both PCI_IRDYn and PCI_TRDYn are asserted. Wait cycles are inserted until both PCI_IRDYn and PCI_TRDYn are asserted. When the L64364 deasserts PCI_IRDYn, it drives it high for one clock cycle and then 3-states it in the following cycles.

PCI_TRDYn **PCI Target Ready** **Bidirectional**

The target, which is the current bus slave, asserts PCI_TRDYn to indicate when there is valid data on PCI_AD[31:0] during a read cycle, or to indicate that it is ready to accept data from PCI_AD[31:0] during a write cycle. The data phase of a transfer completes on any clock cycle when both PCI_IRDYn and PCI_TRDYn are asserted. The L64364 inserts wait cycles until both PCI_IRDYn and PCI_TRDYn are asserted. When the L64364 deasserts PCI_TRDYn, it drives it high for one clock cycle and then 3-states it in the following cycles.

PCI_STOPn **PCI Stop** **Bidirectional**

The target asserts PCI_STOPn to stop the current data transfer. As a master, the L64364 terminates the data transfer when PCI_STOPn is asserted. As a slave, the L64364 asserts PCI_STOPn under the following conditions:

- PCI Slave Write FIFO is full.
- PCI Slave Read FIFO is empty.

When the L64364 deasserts PCI_STOPn, it drives it high for one clock cycle and then 3-states it in the following cycles.

PCI_IDSEL **PCI Initialization Device Select** **Input**

PCI_IDSEL is a chip select which is input to the L64364 during configuration read and configuration write cycles (bus command = 0b1010 or ob1011).

PCI_DEVSELn

PCI Device Select **Bidirectional**

The target, whose address was specified in the address phase of the current transfer, asserts PCI_DEVSELn. As a target, the L64364 asserts PCI_DEVSELn two clock cycles after the PCI Bus master asserts PCI_FRAMEn (medium DEVSEL timing). When the L64364 deasserts PCI_DEVSELn, it drives it high for one clock cycle and then 3-states it in the following cycles.

As a master, the L64364 aborts the transfer if PCI_DEVSELn is not asserted within six clock cycles after PCI_FRAMEn is asserted.

PCI_REQn **PCI Request** **3-State Output**

The L64364 asserts PCI_REQn to request a master transfer on the PCI Bus.

PCI_GNTn **PCI Grant** **Input**

The current bus master asserts PCI_GNTn to indicate to the L64364 when bus ownership is granted. After the L64364 detects PCI_GNTn, it asserts PCI_FRAMEn on the next PCI clock and then initiates the bus transfer.

PCI_PERRn **PCI Parity Error** **Bidirectional**

PCI_PERRn indicates data parity errors have occurred. During a master read or slave write, the L64364 asserts PCI_PERRn within two clock cycles after detecting a parity error. In both cases, it asserts PCI_PERRn for one clock cycle. PCI_PERRn is asserted for more than one clock cycle if multiple parity errors occur on burst transactions. When the L64364 deasserts PCI_PERRn, it drives it high for one clock cycle and then 3-states it during the following cycles.

During a master write, the L64364 checks for parity errors by monitoring PCI_PERRn from the target. If the target asserts PCI_PERRn during a master write, the L64364 interrupts the APU.

PCI_SERRn **PCI System Error** **Open Drain Output**

The L64364 asserts PCI_SERRn when a parity error is detected on PCI_AD[31:0] and PCI_CBE[3:0] during the address phase of a transfer.

PCI_INTn **PCI Interrupt** **Open Drain Output**
L64364 asserts PCI_INTn when the Transmit Messaging Mailbox is not empty and the interrupt is enabled in the Primary Port Control register. Refer to Section 9.3.1, "XPP_Ctrl Register," for additional information.

3.3 Secondary Memory Interface

The Secondary Memory Interface directly controls access to synchronous DRAM, asynchronous SRAM, synchronous SRAM, and some physical layer devices. Refer to Chapter 10 for information about secondary memory configurations and bus functionality.

The interface signals are defined as follows:

SB_D[31:0] **Secondary Data Bus** **Bidirectional**
SB_D[31:0] is a 32-bit wide data bus which provides access to secondary memory. The byte-wide EPROM and all peripheral devices connect to local memory through SB_D[31:24].

SB_A[21:2] **Secondary Bus Address** **Output**
SB_A[21:2] is a 20-bit wide bus for addressing SSRAM, SRAM, SDRAM, EPROM, and peripheral devices. The table below shows the SB_A mapping to these various devices.

SB_A[n]	SSRAM	EPROM	PHY	SDRAM
SB_A[21]	A19	A19	A19	RAS
SB_A[20]	A18	A18	A18	CAS
SB_A[19]	A17	A17	A17	not used
SB_A[18]	A16	A16	A16	not used
SB_A[17]	A15	A15	A15	not used
SB_A[16]	A14	A14	A14	not used
SB_A[15]	A13	A13	A13	A13
SB_A[14]	A12	A12	A12	A12
SB_A[13:2]	A[11:0]	A[11:0]	A[11:0]	A[11:0]

SB_WEn[3:0] **Secondary Bus Write Enables** **Output**
SB_WEn[3:0] are byte write enables for the 32-bit Secondary Bus. The Secondary Bus is defined as little

endian. SB_WEn[3] enables write data on SB_D[31:24], the MSB. SB_WEn[0] enables write data on SB_D[7:0], the LSB. Byte-wide peripheral devices must connect to SB_WEn[0]. Write enables for specific devices are assigned as in the following table.

SB_WEn[n]	SSRAM and 32-Bit EPROM/ SRAM	8-Bit EPROM/ SRAM	Peripheral	SDRAM
SB_WEn[3]	WEn	not used	not used	not used
SB_WEn[2]	WEn	not used	not used	not used
SB_WEn[1]	WEn	not used	not used	not used
SB_WEn[0]	WEn	WEn	WEn	WEn

SB_OEn[3:0] Secondary Bus Output Enables **Output**

SB_OEn[3:0] are the output enables and/or address controls of SSRAMS, SDRAMS, EPROMs, and peripheral devices as defined in the following table.

SB_OEn[n]	SSRAM	EPROM/ SRAM	Peripheral	SDRAM[1]
SB_OEn[3]	not used	not used	not used	DQM (X8), DQMU (X16), DQM[3] (X32)
SB_OEn[2]	not used	not used	not used	DQM (X8), DQML (X16), DQM[2] (X32)
SB_OEn[1]	not used	ALE	ALE	DQM (X8), DQMU (X16), DQM[1] (X32)
SB_OEn[0]	OEn	OEn	OEn	DQM (X8), DQML (X16), DQM[0] (X32)

1. DQM is the LOW-active output enable for byte-wide SDRAM. DQMU and DQML are the LOW-active byte enables for the upper and lower bytes of a 16-bit SDRAM.

SB_CLKO Secondary Bus Clock **Output**

SSRAM, and SDRAM use the SB_CLKO for synchronization. All address, data, and control signals to and from SDRAM and SSRAM are synchronized to the rising edge of SB_CLKO.

SB_PCSn[4:0]

Secondary Bus Page Chip Select **Output**

SB_PCSn[4:0] provide a chip select function for the five pages that are configured in the Secondary Bus control registers. Normally, these signals connect to the chip select inputs of memory devices (either directly or conditioned with address decode). Refer to Section 10.2, "SBC Configuration," for details.

SB_RDYn **Secondary Bus Ready** **Input**

SB_RDYn indicates valid input data is available on SB_D[n]. This asynchronous input may be used instead of the integrated wait state generator to time read and write transfers to asynchronous SRAM, EPROM, and peripheral devices.

SB_RDYn is sampled on the deasserting edge of PCI_RSTn to determine the reset state of SB_Wait fields in the SP_Ctrl Register. See section 10.2.1 on page 10-4. If SB_RDYn is sampled HIGH, then the SB_Wait fields reset to 0xE (14 wait states). If SB_RDYn is sampled LOW, then the SB_Wait fields reset to 0xF (SB_RDYn terminated).

SB_REQn **Secondary Bus Request** **Input**

SB_REQn asserted indicates an external master requests Secondary Bus ownership. It is an asynchronous input.

SB_GNTn **Secondary Bus Grant** **Output**

The L64364 asserts SB_GNTn in response to SB_REQn. Secondary Bus ownership is granted based on a round-robin priority scheme between the PCI slave interface and the L64364 master. All Secondary Bus signals, except SB_CLKO and SB_GNTn, are held 3-stated until SB_REQn is deasserted.

3.4 Utopia Interface

The Utopia Interface conforms to the *ATM Forum's Utopia Level 2 Specification, Version 1.0*. Data and control signals are bidirectional since the L64364 can be configured as either a Utopia master or slave. Utopia interface signals are described in the following paragraphs.

TX_DATA[7:0]

Transmit Cell Data **Bidirectional**
Master Mode (output)

TX_DATA[7:0] is the Utopia transmit cell data output from the L64364 to one of the external Physical Layer (PHY) devices. Transmit cells are output from the transmit FIFO of the L64364.

Slave Mode (input)

TX_DATA[7:0] is the Utopia transmit cell data input from an external Utopia master device. Cell data is input to the receive FIFO of the L64364.

TX_SOC **Transmit Start of Cell** **Bidirectional**
Master Mode (output)

TX_SOC is output from the L64364 to indicate the first byte of a cell. This signal pulses HIGH for one Utopia transmit-clock cycle at the first byte of the cell header when the cell length is 52 or 53 bytes. TX_SOC pulses HIGH for one Utopia transmit-clock cycle at the first byte of the prepended routing tag when the cell length exceeds 53 bytes.

Slave Mode (input)

TX_SOC is input to the L64364 from the Utopia master to indicate the first byte of a cell. A HIGH on this signal when sampled with respect to the Utopia transmit clock indicates the first byte of the cell header for 52–53 byte cells or the first byte of the routing tag for 56–65 byte cells.

TX_ENBn **Transmit Enable** **Bidirectional**
Master Mode (output)

The L64364 asserts TX_ENBn to designate valid transmit cell data on TX_DATA[7:0].

Slave Mode (input)

TX_ENBn is asserted by the Utopia master to the L64364 to designate valid transmit cell data on TX_DATA[7:0]. Transmit cell data is sampled on the rising edge of TX_CLK when TX_ENBn is asserted.

TX_CLK **Transmit Clock** **Input**

TX_CLK is the ATM layer interface clock used to synchronize transmit cell data transfers. TX_CLK has a maximum frequency of 50 MHz and is an input that is independent of master or slave configuration.

TX_PRTY **Transmit Parity** **Bidirectional**

Master Mode (output)

TX_PRTY is set to logic 1 to signal odd parity over TX_PRTY and TX_DATA[7:0] if the total number of 1's in TX_DATA[7:0] is even. It is reset to logic 0 if the total number of 1's in TX_DATA[7:0] is odd.

Slave Mode (input)

TX_PRTY is set or reset by the Utopia master to designate odd parity on TX_DATA[7:0] as described for master mode. If parity is enabled, the L64364 will interrupt the APU to handle errored cells.

TX_ADDR[4:0]

Transmit Address **Bidirectional**

Master Mode (output)

TX_ADDR[4:0] is output to select the PHY device that is the destination for the next transmit cell. The L64364 supports up to 24 PHY devices 0–23.

Slave Mode (input)

During polling by the Utopia master, the L64364 monitors TX_ADDR[4:0] for its PHY device address, which is programmed in the ACI Control register. The PHY device address range is 0–30 in slave mode. L64364 asserts TX_CLAV[0] whenever the Receive FIFO can accept a transmit cell from the Utopia master and TX_ADDR[4:0] matches its programmed PHY device address.

TX_CLAV[3:0]

Transmit Cell Available **Bidirectional**

Master Mode (input)

The L64364 monitors the TX_CLAV[3:0] signals to determine if the target PHY device is capable of accepting the next cell from the transmit FIFO. When TX_CLAV is asserted, the transmit cell is transferred to the PHY device on TX_DATA[7:0]. If TX_CLAV is not asserted prior to the expiration of the L64364's programmable

watchdog timer, the transmit cell is moved to an Error FIFO and the APU is optionally interrupted for error handling.

TX_CLAV definition is dependent on whether direct or multiplexed polling is selected. In direct polling, up to four PHY devices are supported and each one connects to one of the TX_CLAV[3:0] inputs. Multiplexed polling uses only the TX_CLAV[0] input. Up to 24 PHY devices will then connect their TxCLAV outputs to TX_CLAV[0].

Slave Mode (output)

The L64364, when selected by the Utopia master, asserts TX_CLAV[0] if the Receive FIFO has room for at least one cell. TX_CLAV[3:1] are not used.

RX_DATA[7:0]

Receive Cell Data **Bidirectional**
Master Mode (input)

RX_DATA[7:0] is the Utopia receive cell data input from the PHY devices. Receive cell data is destined for the receive FIFO of the L64364.

Slave Mode (output)

RX_DATA[7:0] is the Utopia receive cell data output from the transmit FIFO of the L64364 to the Utopia master.

RX_SOC **Receive Start of Cell** **Bidirectional**
Master Mode (input)

RX_SOC is asserted to the L64364 at the first byte of the cell. This signal pulses HIGH for one Utopia receive-clock cycle at the first byte of the cell header when the cell length is 52 or 53 bytes. RX_SOC pulses HIGH for one Utopia receive-clock cycle at the first byte of the prepended routing tag when the cell length exceeds 53 bytes.

Slave Mode (output)

The L64364 asserts RX_SOC to indicate the first byte of a cell. A HIGH on this signal when sampled with respect to the Utopia receive clock indicates the first byte of the cell header for 52–53 byte cells or the first byte of the routing tag for 56–65 byte cells.

RX_ENBn **Receive Enable** **Bidirectional**
Master Mode (output)

After a PHY device is selected, the L64364 asserts RX_ENBn to indicate it is ready to accept receive cell data. In a single-PHY application, RX_ENBn remains asserted as long as the receive FIFO is not full and the PHY device has cells available. This enables back-to-back cell reception. In a multi-PHY application, RX_ENBn remains asserted as long as the receive FIFO is not full and a higher priority PHY device does not assert its CLAV signal.

Slave Mode (input)

The L64364 monitors the RX_ENBn input to determine when the Utopia master is ready to receive cells from the transmit FIFO. When selected by the Utopia master, the L64364 uses RX_ENBn to enable the 3-state drivers on RX_DATA[7:0] and RX_SOC, and then initiate the Utopia receive cell transfer.

RX_CLK **Receive Clock** **Input**
RX_CLK is the ATM layer interface clock used to synchronize receive cell data transfers. RX_CLK has a maximum frequency of 50 MHz and is an input that is independent of master or slave configurations.

RX_PRTY **Receive Odd Parity** **Bidirectional**
Master Mode (input)

RX_PRTY is input to the L64364 by the PHY device to designate odd parity. It is set to logic 1 when the total number of 1's in RX_DATA[7:0] is even and reset to 0 if the number of 1's in RX_DATA[7:0] is odd. If parity is enabled, the L64364 interrupts the APU to handle errored cells.

Slave Mode (output)

RX_PRTY is set or reset by the L64364 to designate odd parity as described for master mode.

RX_ADDR[4:0]
Receive Address **Bidirectional**
Master Mode (output)

The L64364 places the address of the PHY device being polled on RX_ADDR[4:0]. The L64364 supports up to 24 PHY devices 0–23.

Slave Mode (input)

During polling by the Utopia master, the L64364 monitors RX_ADDR[4:0] for its PHY device address which is programmed in the ACI Control register. The PHY device address range is 0–30 in slave mode. When RX_ADDR[4:0] matches the programmed PHY device address, the L64364 asserts RX_CLAV[0] if the transmit FIFO is not empty. The L64364 also asserts RX_CLAV[0] if the PHY device address is zero and idle cell generation is enabled.

RX_CLAV[3:0]

Receive Cell Available **Bidirectional**

Master Mode (input)

The L64364 monitors RX_CLAV[3:0] signals to determine if the target PHY device has a receive cell available. When the target PHY device asserts RX_CLAV, the L64364 asserts RX_ENBn to enable the receive cell transfer.

The definition of RX_CLAV depends on whether direct or multiplexed polling is selected. In direct polling, up to four PHY devices are supported and each one connects to one of the RX_CLAV[3:0] inputs. Multiplexed polling uses only the RX_CLAV[0] input. Up to 24 PHY devices can then connect their RxCLAV outputs to RX_CLAV[0].

Slave Mode (output)

The L64364, when selected by the Utopia master, asserts RX_CLAV[0] if the transmit FIFO is not empty. RX_CLAV[3:1] are not used.

3.5 Clocks and Utility Signals

This section defines the clocks, clock select signals, and miscellaneous control signals. Refer to Chapter 11 for detailed information on clocks and clock control signals.

SYS_CLK_PCI

PCI Clock Select **Input**

SYS_CLK_PCI, when HIGH, selects PCI_CLK as the system clock source for the L64364. When SYS_CLK_PCI is LOW, SYS_CLK is selected as the system clock source.

SYS_CLK_PCI must be tied to power or ground and must not change during the operation of the L64364.

SYS_CLK **System Clock** **Input**
SYS_CLK is the system clock input for the L64364. The maximum clock frequency is for PX80 is 80 MHz and for PX100 is 100 MHz.

SYS_PLL **PLL Select** **Input**
When SYS_PLL is HIGH, the clock selected by SYS_CLK_PCI is multiplied by two in the internal phase-locked loop before being used as the internal system clock. When SYS_PLL is LOW, the selected clock is used at its frequency. This signal must be tied to power or ground and must not change during the operation of the L64364 operation.

PLL_IDDTn **PLL Operations** **Input**
PLL_IDDTn enables manufacturing test of the internal phase-locked loop. This pin must be tied to logic ground (VSS) for normal operation.

PLL_LP2 **PLL Loop Filter Input** **Input**
The PLL_LP2 pin is a tap off the line connecting the PLL's phase detector charge pump output and VCO input. An external loop filter, consisting of one resistor and two capacitors, must be connected between this pin and the PLLAGND pin. See Figure 11.2 for configuration details.

PLLVSS **PLL Ground**
The PLL uses one reference ground pin isolated from all other grounds.

PLLVDD **PLL Power** **Input**
The PLL uses one 3 V power pin isolated from all other power lines.

PLLAGND **PLL Analog Ground**
A separate reference ground pin isolated from all other grounds is provided for the analog section of the PLL.

TM_CLK **Timer Reference Clock** **Input**
The TM_CLK input may be used as a reference clock for the seven general-purpose timers or the Time Stamp Counter. TM_CLK has a maximum frequency of one-third the internal system clock rate and may be asynchronous

to the system clock. Refer to Section 8.2, “Timer Clock Selection,” for additional information.

SYS_OE **System Output Enable** **Input**
SYS_OE, when deasserted, 3-states all outputs of the L64364.

3.6 APU Signals

The section summarizes the ATM Processing Unit’s (APU) external signals.

SYS_NMIn **System Nonmaskable Interrupt** **Input**
This signal provides a nonmaskable interrupt to the APU. When asserted, it causes the APU to unconditionally execute the nonmaskable interrupt handler.

SYS_INTn[1:0]
System Interrupt **Inputs**
If vectored interrupts to the APU are enabled, SYS_INTn[1] and SYS_INTn[0] cause vectored interrupts 5 and 4, respectively. SYS_INTn[1:0] are asynchronous, level-sensitive inputs. They are resynchronized to the L64364's system clock. SYS_INTn[1:0] must remain asserted until the associated interrupt is acknowledged by the interrupt service routine executed by the APU. For information about vectored interrupts, their sources, and how they are enabled, refer to Section 4.8, “Interrupts.”

SYS_BOOT[1:0]
System Boot Source **Inputs**
SYS_BOOT[1:0] select the boot source according to the table below. SYS_BOOT[1:0] are sampled on the rising edge (deassertion) of PCI_RSTn. Refer to Section 4.11, “Boot Procedures,” for further information.

SYS_BOOT[1:0]	Boot Source
0b00	Secondary Bus EPROM
0b01	Not Used
0b10	Cell Buffer Memory
0b11	Serial EPROM

SYS_PSTALLn

System Pipeline Stalled **Output**

SYS_PSTALLn is asserted when the APU internal pipeline is stalled. This signal is for system performance tuning. Refer to Section 4.2.3, "CW4011 Pipeline," for information about APU pipeline functionality and performance considerations.

SYS_CPCOND

System Coprocessor Condition Code Bit **Input**

This signal sets the state of the APU's Coprocessor Condition Code bit 0. This allows the APU to directly test the status of external logic using Branch or Coprocessor Condition instructions.

3.7 Serial EPROM Interface

The Serial EPROM Interface provides an optional APU boot source. This section describes the Serial EPROM Interface signals.

SE_CLK **Serial EPROM Clock** **Output**

SE_CLK is the clock used to load boot code from the serial EPROM. The SE_CLK frequency is the internal system clock frequency divided by 32. For example, if the L64364 system clock is 66 MHz, the SE_CLK will be 2.06 MHz.

SE_DI **Serial EPROM Data Input** **Input**

SE_DI is the boot code from the serial EPROM. The code is packed so the first bit out of the serial EPROM is the least significant bit of byte 0 (the most significant byte). Data on SE_DI is sampled on the rising edge of SE_CLK.

SE_ACK **Serial EPROM Acknowledge** **Input**

External logic asserts this signal to indicate when there is valid serial EPROM data present on SE_DI. When the external logic asserts SE_ACK, the L64364 captures the data present on SE_DI on the rising edge of SE_CLK.

3.8 JTAG Test Interface

The JTAG Test Interface conforms to *IEEE 1149.1, Standard Test Access Port and Boundary Scan Architecture*. The interface signals are described in this section.

JTAG_TCLK **JTAG Test Clock** **Input**
JTAG_TCLK shifts the boundary scan register and has a maximum frequency of 20 MHz.

JTAG_TDI **JTAG Test Data Input** **Input**
The test data input, JTAG_TDI, is sampled on the rising edge of JTAG_TCLK.

JTAG_TDO **JTAG Test Data Output** **Output**
JTAG_TDO is the test data output of the boundary scan register. The output data is synchronized to the falling edge of JTAG_TCLK.

JTAG_TM **JTAG Test Mode** **Input**
JTAG_TM is the control input to the JTAG Test Access Port controller.

JTAG_TRSTn **JTAG Test Reset** **Input**
JTAG_TRSTn, when asserted, resets the JTAG Test Access Port controller.

TEST_EN **Factory Test Enable** **Input**
TEST_EN, when asserted, enables factory test. This signal must be tied to ground during normal operation.

SCAN_EN **Factory Scan Test Enable** **Input**
SCAN_EN, when asserted, enables factory scan test chains. This signal must be tied to ground during normal operation.

3.9 Power and Ground Pins

This section lists the power and ground pins. The number of power and ground pins has been selected to provide reliable operation and HIGH I/O signal integrity.

VSS **Logic Ground**
Thirteen pins are dedicated to reference ground for internal chip logic.

VDD **Logic Power** **Input**
Twelve pins are dedicated to the 3.3 V supply voltage for internal chip logic.

VSS2 **I/O Ground**
Nine pins are dedicated to reference ground for the I/O pad ring.

VDD2 **I/O Power**
Nine pins are dedicated to powering the I/O pad ring. Nominal voltage is 3.3 V.

VCC (+ 5 V) **PCI Clamp Voltage** **Input**
Seven pins provide voltage to the PCI input clamp diodes. This voltage is the required 11 V overvoltage specification of the *PCI Local Bus Specification 2.1, Section 4.2.1.3*. No power dissipation is associated with these pins. These pins sink current when the PCI inputs are raised above 5 V.

Chapter 4
ATM Processing Unit

This chapter describes the ATM Processing Unit (APU) and contains the following sections:

- Section 4.1, “APU Overview,” page 4-1
- Section 4.2, “APU Architecture,” page 4-3
- Section 4.3, “APU Instruction Set Summary,” page 4-10
- Section 4.4, “CP0 Data Manipulation Registers,” page 4-49
- Section 4.5, “Cache Memory,” page 4-50
- Section 4.6, “Exceptions,” page 4-59
- Section 4.7, “Memory Map,” page 4-95
- Section 4.8, “Interrupts,” page 4-101
- Section 4.9, “CW4011 OCA Bus Accesses,” page 4-112
- Section 4.10, “Bus Watchdog Timers,” page 4-112
- Section 4.11, “Boot Procedures,” page 4-118

Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” should not be modified.

4.1 APU Overview

The APU is built on rgw LSI Logic MiniRISC CW4011 Superscalar Microprocessor Core. The CW4011 is a member of the LSI Logic MiniRISC family, the next generation of MIPS RISC products. LSI Logic offers the CW4011 as a CoreWare® product for use in customer ASIC designs. LSI Logic also embeds the CW4011 in Application-Specific Standard Products (ASSPs), such as the L64364 ATMizer II+ chip. Refer

to the *MiniRISC CW4011 Superscalar Microprocessor Core Technical Manual*, LSI Logic Order No. C14040, for information about the CW4011 not covered in this chapter.

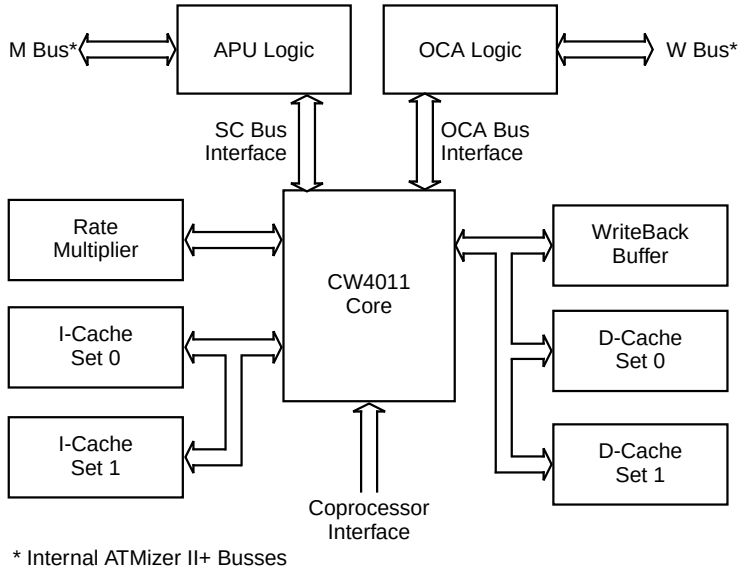
4.1.1 Block Diagram

The APU block diagram is shown in Figure 4.1. In addition to the CW4011 core, the APU contains the following building blocks:

- Direct-mapped or two-way set associative instruction cache (8 Kbytes)
- Direct-mapped or two-way set associative data cache (4 Kbytes)
- A write back buffer for write back cache mode
- A floating-point multiplier unit for ATM ABR rate calculations

The APU also includes logic that allows initial program loading from a serial EPROM, a byte-wide EPROM, or a host download to the ATMizer CBM.

Figure 4.1 APU Block Diagram



4.1.2 Features

The APU has the following key features:

- Full MIPS II instruction set implementation (R4000 32-bit mode compatible).
- Instruction set extensions to support ATM ABR calculations.
- Superscalar execution: two instructions per clock cycle.
- 32-bit memory and 64-bit cache interface.
- PX80 delivers 150 Dhrystone MIPS at 80 MHz and PX100 delivers 188 Dhrystone MIPS at 100 MHz.
- PX80, 160 native MIPS peak (110 native MIPS sustained) with standard compiled MIPS code at 80 MHz and PX100, 200 native MIPS peak (138 MIPS sustained) with standard compiled MIPS code at 100 MHz.
- Integrated cache controllers with separate 8 Kbyte instruction and 4 Kbyte data cache.

4.2 APU Architecture

The APU is fully compatibility with both the R3000 and R4000 32-bit instruction sets (MIPS I and MIPS II) but uses an updated hardware architecture to provide higher absolute performance than any other available MIPS solution. The APU also provides substantially better instructions-per-clock performance than other MIPS processors. At the same time, the hardware design remains compact compared to other superscalar architectures.

The APU can issue and retire two instructions per cycle using a combination of four independent execution units:

- The Arithmetic Logic Unit (ALU)
- The Load/Store/Add Unit (LSU)
- The Branch Unit
- The Rate Multiply Unit

The LSU can execute add and load immediate instructions (in addition to load and store), making it possible for the L64364 to perform an add instruction at the same time it executes another add or logical instruction.

All instructions, except multiply and divide, can be completed in a single cycle.

Load instructions have a single hardware delay slot for loads that hit in the cache, but the hardware interlocks on register conflicts so that a `NOP` is required in the delay slot. On a load miss, the APU extends the hardware conflict detection so that, if the load data is not required by subsequent instructions in the pipeline, the CPU is not stalled. The operation is called load scheduling. The APU supports store instructions with both a write back cache and a write buffer.

The APU has an instruction prefetch queue and branch prediction logic to boost branch performance so that correctly predicted branches are retired with no penalty, and incorrectly predicted branches normally have a penalty of just one cycle. The APU accomplishes branch prediction with a simple hardware algorithm that has an accuracy of greater than 90% for most application code.

4.2.1 CW4011 Core

Figure 4.2 shows a block diagram of the CW4011 core. It includes the following blocks:

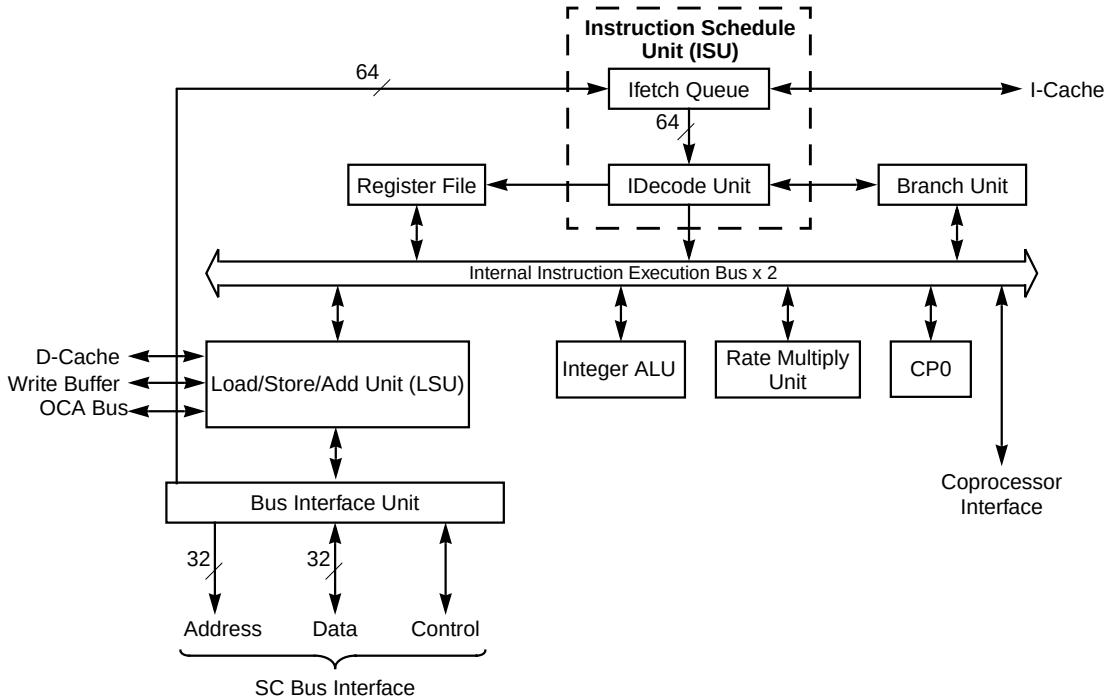
- Ifetch Queue
- IDecode Unit
- Branch Unit
- Register File
- Load/Store/Add Unit (LSU)
- Integer Arithmetic Logic Unit (ALU)
- Floating-Point Rate Multiply Unit
- Bus Interface Unit (BIU)
- SC Bus Interface
- Coprocessor Interface
- System Control Coprocessor (CP0)

The Ifetch Queue optimizes the supply of instructions to the microprocessor, even across breaks (jumps and branches) in the sequential flow of execution. The IDecode Unit decodes the instructions from the Ifetch Queue; determines the actions required for the instruction execution; and manages the Register File (RFile), LSU, ALU, and Multiplier Units accordingly.

The Branch Unit is used when branch and jump instructions are recognized within the instruction stream.

The Register File contains the APU's general purpose registers. It supplies source operands to the execution units and handles the storage of results to target registers.

Figure 4.2 CW4011 Block Diagram



Three units perform logical, arithmetic, and data-movement operations. The Load/Store/Add Unit (LSU) manages loads and stores of data values. Loads come from either the D-Cache or the SC Bus Interface in the event of a D-Cache miss. Stores pass to the D-Cache and the

SC Bus Interface through the Write Buffer. The LSU also performs a restricted set of arithmetic operations, including the addition of an immediate offset as required in address calculations. The Integer ALU Unit calculates the result of an arithmetic or logical operation. The Rate Multiply Unit performs integer multiply/divide operations and floating point operations in the format adopted by the ATM Forum for ABR calculations.

The Bus Interface Unit manages the flow of instructions and data between the CW4011 core and the system using the SC Bus Interface. The SC Bus Interface provides the main channel for communication between the CW4011 core and the other functional blocks in the ATMizer II+ chip.

The Coprocessor Interface provides the status of key ATMizer II+ resources. Coprocessor Condition codes are set for the following:

- CPCond3: EDMA RxCell Request Queue full
- CPCond2: EDMA TxCell Request Queue full
- CPCond1: EDMA Buffer Request Queue full
- CPCond0: SYS_CPCOND input to L64364 asserted

These conditions are used by the Branch on Coprocessor (`BCzT` and `BCzF`) instructions described in Section 4.3.7, “Coprocessor Instructions.” `CPCond0` interrupts the CW4011.

4.2.2 Cache and External Interface

The instruction cache and controller are an integral part of the CW4011 Instruction Schedule Unit (ISU), and the data cache and controller are an integral part of the LSU. The Write Buffer is also part of the LSU.

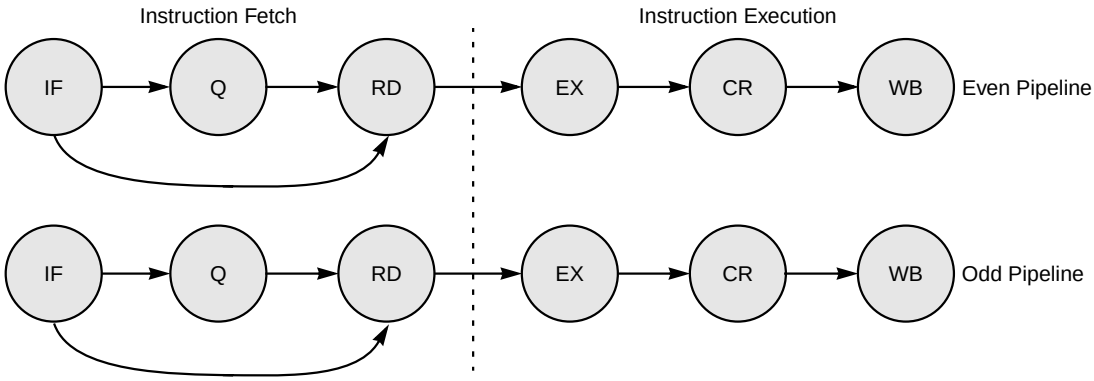
In addition, a simple memory interface unit, the BIU, provides the interface to Primary Memory, Secondary Memory, and the ATMizer’s Cell Buffer Memory and registers. The BIU presents a nonmultiplexed interface with a 32-bit data bus and 32-bit address bus. A write back buffer is provided to support write back mode to the data cache.

4.2.3 CW4011 Pipeline

As shown in Figure 4.3, the superscalar CW4011 has two, concurrent, six-stage pipelines or slots—an even and an odd. The first three stages

are labelled the instruction fetch phase, and the last three stages are labelled the instruction execution phase.

Figure 4.3 CW4011 Instruction Pipeline



In general, the execution of a single CW4011 instruction consists of the following stages:

1. IF (Instruction Fetch) – The CW4011 fetches the instruction during this stage.
2. Q (Queuing) – Instructions may enter this conditional stage if they deal with branches or register conflicts. An instruction that does not cause a branch or register conflict is fed directly to the RD stage.
3. RD (Read) – During this stage, any required operands are read from the Register File while the instruction is decoded.
4. EX (Execute) – All instructions are executed in this stage. In addition, conditional branches are resolved and load/store address calculations are performed during this stage.
5. CR (Cache Read) – In this stage, cache read operations are performed for load and store instructions. Data is returned to the register bypass logic at the end of the CR stage.
6. WB (Write Back) – Results are written into the Register File during this stage.

Once it has accepted an instruction from the previous stage, each stage holds the instruction for re-execution if the pipeline stalls.

4.2.3.1 Instruction Fetch and Scheduling Stages

The IF, Q, and RD stages fetch and issue two instructions per cycle to the execute stage. For simplicity, the CW4011 fetches instructions as doubleword aligned pairs (slot 0 and slot 1). In the instruction decode stage, there is a two-instruction window. When only slot 0 can be scheduled because slot 1 has a dependency, the window slides down one instruction. In other words, although instructions are always fetched as doubleword pairs, they are scheduled on single-word boundaries.

The Q stage executes branch instructions with minimal penalty. In general, the CW4011 fills the Q stage whenever the RD stage has to stall, which occurs fairly often for register conflicts, cache misses, and resource conflicts. Filling the Q stage in this case allows the IF stage to work ahead one cycle.

When a branch instruction type is encountered and the Q stage is active, the branch is predicted to be taken and instruction fetching starts at the branch address. At this point, the Q stage holds the next nonbranch instructions to be executed. The branch target enters the RD stage, bypassing the Q stage. When the branch instruction enters the Execute stage, the branch condition is resolved. If the branch was correctly predicted, the instructions in the Q stage are cancelled. If the branch was incorrectly predicted, the branch target is cancelled. In this later case, the nonbranch sequential instructions are taken from the Q stage, and the IF stage is restarted at the nonbranch sequential stream. A different case occurs when the branch instruction is in the odd instruction slot.

In general, a branch instruction that is correctly predicted from the even slot with the Q stage full has no cycle penalty associated with it. In the case where the branch is incorrectly predicted, the branch has a one-cycle penalty.

If the branch instruction was in the odd slot, the branch delay slot instruction always executes by itself and has no chance to fill the other execution slot. For that reason, it may be advantageous if the software assembler places branches in even word addresses.

The branch prediction logic must be capable of looking at two instructions at a time, from either the Q latches or the RD latches, depending on whether the Q stage is active. If one of the instructions is a branch, the offset in that instruction is passed into a dedicated adder

to calculate the branch address for the IF stage instruction fetch. Because this is done speculatively, the nonbranch value of the Program Counter (PC) is also saved for the possible restart of the sequential instructions from the Q stage.

After an instruction pair passes into the RD stage, it is decoded and, at the same time, the register source addresses are passed to the register file so the operands can be read. Register dependencies and resource dependencies are checked in this stage. If the instruction in slot 0 has no dependency on a register or resource currently tied up by a previous instruction, it is passed immediately into the EX stage where it forks to the appropriate execution unit. The instruction in slot 1 may also be dependent on a resource or register in slot 0, so it must be checked for dependencies against both slot 0 and any previous unretired instruction. If either instruction must be held in the RD stage and the Q stage is not full, the IF stage is allowed to continue in order to fill the Q stage. If the Q stage is full, then the Q and IF stages are frozen (stalled).

In the RD stage, register bypass opportunities are considered and the bypass multiplexers control signals are set for potential bypass cases from a previous instruction still in the pipeline.

4.2.3.2 Execute Stage

During instruction execution, a pair of instructions (or a single instruction when there was a previous block) are individually passed to independent execution units. Each execution unit receives its operands from the register bypass logic and an instruction from the instruction scheduler. Each instruction spends one run cycle in an execution unit. For ALU and other single cycle instructions, the result is then fed to the register/bypass unit for the CR stage.

4.2.3.3 Cache Read and Write Back Stages

For load and store instructions, the cache lookup occurs during the CR stage. For load instructions, data is returned to the register/bypass unit during the CR stage, including data loads to Coprocessor 0.

For all other instructions, CR and WB are holding stages used to hold the result of the execute stage for write back to the register file.

4.3 APU Instruction Set Summary

The CW4011 instruction set with ATMizer II+ specific extensions is summarized in this section. Table 4.1 summarizes the instruction set for the APU. The APU supports both MIPS I and MIPS II instructions, and also implements some additional CW4011-specific and APU-specific instructions. All instructions are 32-bits long.

Table 4.1 APU Instruction Set Summary

Op	Description	Op	Description
Arithmetic Instructions: ALU Immediate			
ADDI	Add Immediate	ANDI	AND Immediate
ADDIU	Add Immediate Unsigned	ORI	OR Immediate
SLTI	Set on Less Than Immediate	XORI	Exclusive OR Immediate
SLTIU	Set on Less Than Immediate Unsigned	LUI	Load Upper Immediate
Arithmetic Instructions: Three-Operand, Register-Type			
ADD	Add	SLTU	Set on Less Than Unsigned
ADDU	Add Unsigned	AND	AND
SUB	Subtract	OR	OR
SUBU	Subtract Unsigned	XOR	Exclusive OR
SLT	Set on Less Than	NOR	NOR
Branch Likely Instructions[1]			
BEQL	Branch on Equal Likely	BGTZL	Branch on Greater Than Zero Likely
BNEL	Branch on Not Equal Likely	BLTZL	Branch on Less Than Zero Likely
BLEZL	Branch on Less than or Equal to Zero Likely	BLTZALL	Branch on Less Than Zero And Link Likely
BGEZL	Branch on Greater than or Equal to Zero Likely	BGEZALL	Branch on Greater than or Equal to Zero and Link Likely

(Sheet 1 of 4)

Table 4.1 APU Instruction Set Summary (Cont.)

Op	Description	Op	Description
Cache Maintenance			
FLUSHI	Flush I-Cache	FLUSHID	Flush I-Cache and D-Cache
FLUSHD	Flush D-Cache		
Coprocessor Instructions			
BCzT (BCzTL)	Branch on Coprocessor z True (Likely)	BCzF (BCzFL)	Branch on Coprocessor z False (Likely)
Jump and Branch Instructions			
J	Jump	BLEZ	Branch on Less than or Equal to Zero
JAL	Jump And Link	BGTZ	Branch on Greater Than Zero
JR	Jump Register	BLTZ	Branch on Less Than Zero
JALR	Jump And Link Register	BGEZ	Branch on Greater than or Equal to Zero
BEQ	Branch on Equal	BLTZAL	Branch on Less Than Zero And Link
BNE	Branch on Not Equal	BGEZAL	Branch on Greater than or Equal to Zero And Link
Load/Store Instructions			
LB	Load Byte	SH	Store Halfword
LBU	Load Byte Unsigned	SW	Store Word
LH	Load Halfword	SWL	Store Word Left
LHU	Load Halfword Unsigned	SWR	Store Word Right
LW	Load Word	LL[1]	Load Linked
LWL	Load Word Left	SC[1]	Store Conditional
LWR	Load Word Right	SYNC[1]	Sync
SB	Store Byte		

(Sheet 2 of 4)

Table 4.1 APU Instruction Set Summary (Cont.)

Op	Description	Op	Description
Multiply/Divide Instructions			
MULT	Multiply	MFHI	Move From HI
MULTU	Multiply Unsigned	MFLO	Move From LO
DIV	Divide	MTHI	Move To HI
DIVU	Divide Unsigned	MTLO	Move To LO
Other Computational Instructions[2]			
ADDCIU	Add Circular Immediate	SELSL	Select and Shift Left
FFS	Find First Set	MIN	Minimum
FFC	Find First Clear	MAX	Maximum
SELSR	Select and Shift Right	SELRR	Select and Rotate Right
Rate Instructions[3]			
RMUL	Rate Multiply	R2U	Rate to Integer
RADD	Rate Add	U2R	Integer to Rate
RSUB	Rate Subtract		
Shift Instructions			
SLL	Shift Left Logical	SLLV	Shift Left Logical Variable
SRL	Shift Right Logical	SRLV	Shift Right Logical Variable
SRA	Shift Right Arithmetic	SRAV	Shift Right Arithmetic Variable
Special Instructions			
SYSCALL	System Call	BREAK	Breakpoint

(Sheet 3 of 4)

Table 4.1 APU Instruction Set Summary (Cont.)

Op	Description	Op	Description
System Control Coprocessor (CP0) Instructions			
MTC0	Move To CP0	RFE	Restore From Exception
MFC0	Move From CP0	WAITI[2]	Wait for Interrupt
ERET	Exception Return		
Trap Instructions[1]			
TEQ	Trap on Equal	TLT	Trap on Less Than
TEQI	Trap on Equal Immediate	TLTI	Trap on Less Than Immediate
TGE	Trap on Greater than or Equal	TLTU	Trap on Less Than Unsigned
TGEI	Trap on Greater than or Equal Immediate	TLTIU	Trap on Less Than Immediate Unsigned
TGEU	Trap on Greater than or Equal Unsigned	TNE	Trap If Not Equal
TGEIU	Trap on Greater than or Equal Immediate Unsigned	TNEI	Trap If Not Equal Immediate
(Sheet 4 of 4)			

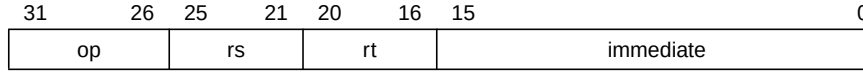
1. MIPS II Instructions.
2. CW4011-specific instructions.
3. APU-specific instructions.

4.3.1 Instruction Set Formats

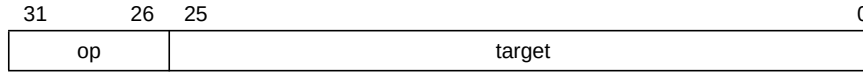
Every instruction consists of a single word (32-bits) aligned on a word boundary. As shown in Figure 4.4, there are three instruction formats: I-type (immediate), J-type (jump), and R-type (register). The restricted format approach simplifies instruction decoding. The compiler and assembler can synthesize more complicated (and less frequently used) operations and addressing modes.

Figure 4.4 Instruction Formats

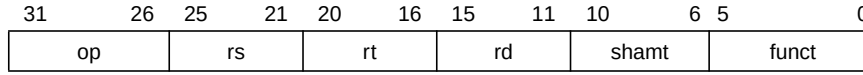
I-Type (Immediate)



J-Type (Jump)



R-Type (Register)



op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination register)
immediate	16-bit immediate, branch displacement, or address displacement
target	26-bit jump target address
rd	5-bit destination register specifier
shamt	5-bit shift amount
funct	6-bit function field

4.3.2 Load and Store Instructions

The load and store instructions summarized in Table 4.2 are all I-type instructions and move data between memory and general registers. The only addressing mode directly supported in the base architecture is base register plus 16-bit signed immediate offset.

The MIPS II extensions add the Load Linked and Store Conditional instructions which support multiple processors, and the Sync instruction which synchronizes loads and stores. The APU supports these instructions. They are summarized in Table 4.3.

The load/store instruction operation code (opcode) determines the access type, which in turn indicates the size of the data item to be loaded or stored. The bytes used within the addressed word can be determined directly from the access type and the two low-order bits of the address as shown in Figure 4.5. Note that certain combinations of access type and low-order address bits can never occur; only the combinations shown in Figure 4.5 are permissible.

Figure 4.5 Byte Specifications for Loads/Stores

Access Type	Low-Order Address Bits: A1	A0	Bytes Accessed (Big Endian, 31 … 0)
Word	0	0	0 1 2 3
Tribyte	0	0	0 1 2
	0	1	1 2 3
Halfword	0	0	0 1
	1	0	2 3
Byte	0	0	0
	0	1	1
	1	0	2
	1	1	3

Table 4.2 Load and Store Instructions Summary

Instruction	Format and Description
Load Byte	`LB rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Sign-extend contents of addressed byte and load into rt.
Load Byte Unsigned	`LBU rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Zero-extend contents of addressed byte and load into rt.
Load Halfword	`LH rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Sign-extend contents of addressed halfword and load into rt.
Load Halfword Unsigned	`LHU rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Zero-extend contents of addressed halfword and load into rt.
Load Word	`LW rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address, and load the addressed word into rt.
Load Word Left	`LWL rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Shift addressed word left so that addressed byte is leftmost byte of a word. Merge bytes from memory with contents of register rt and load result into register rt.
Load Word Right	`LWR rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Shift addressed word right so that addressed byte is rightmost byte of a word. Merge bytes from memory with contents of register rt and load result into register rt.
Store Byte	`SB rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Store least-significant byte of register rt at addressed location.
Store Halfword	`SH rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Store least-significant halfword of register rt at addressed location.
Store Word	`SW rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Store contents of register rt at addressed location.

(Sheet 1 of 2)

Table 4.2 Load and Store Instructions Summary (Cont.)

Instruction	Format and Description
Store Word Left	`SWL rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Shift contents of register rt right so that the leftmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.
Store Word Right	`SWR rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Shift contents of register rt left so that the rightmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.
(Sheet 2 of 2)	

Table 4.3 Load and Store Instruction Summary–MIPS II ISA Extensions

Instruction	Format and Description
Load Linked	`LL rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Sign-extend contents of addressed word and load into register rt.
Store Conditional	`SC rt, offset(base)` Sign-extend 16-bit `offset` and add to contents of register base to form address. Conditionally store low-order word of register rt at address, based on whether the load-link has been “broken.”
Sync	`SYNC` Complete all outstanding load and store instructions before allowing any new load or store instruction to start.

4.3.3 Computational Instructions

Computational instructions perform arithmetic, logical, and shift operations on values in registers. Computational instructions occur in both R-type (both operands are registers) and I-type (one operand is a 16-bit immediate) formats. There are six categories of computational instructions:

- ALU Immediate instructions (see Table 4.4).
- 3-Operand, Register-Type instructions (see Table 4.5).
- Shift instructions (see Table 4.6).

- Multiply/Divide instructions (see Table 4.7).
- Computational CW4011 Instruction Extensions (see Table 4.10).
- APU Rate Instruction Extensions (see Table 4.11).

Table 4.4 ALU Immediate Instruction Summary

Instruction	Format and Description
Add Immediate	`ADDI rt, rs, immediate` Add 16-bit, sign-extended `immediate` to register rs and place 32-bit result in register rt. Trap on two's complement overflow.
Add Immediate Unsigned	`ADDIU rt, rs, immediate` Add 16-bit, sign-extended `immediate` to register rs and place 32-bit result in register rt. Do not trap on overflow.
Set on Less Than Immediate	`SLTI rt, rs, immediate` Compare 16-bit, sign-extended `immediate` with register rs as signed 32-bit integers. Result = 1 if rs is less than immediate; otherwise result = 0. Place result in register rt.
Set on Less Than Immediate Unsigned	`SLTIU rt, rs, immediate` Compare 16-bit, sign-extended `immediate` with register rs as unsigned 32-bit integers. Result = 1 if rs is less than immediate; otherwise result = 0. Place result in register rt.
AND Immediate	`ANDI rt, rs, immediate` Zero-extend 16-bit `immediate`, AND with contents of register rs, and place result in register rt.
OR Immediate	`ORI rt, rs, immediate` Zero-extend 16-bit `immediate`, `OR` with contents of register rs, and place result in register rt.
Exclusive OR Immediate	`XORI rt, rs, immediate` Zero-extend 16-bit `immediate`, exclusive `OR` with contents of register rs, and place result in register rt.
Load Upper Immediate	`LUI rt, immediate` Shift 16-bit `immediate` left 16-bits. Set least-significant 16-bits of word to zeros. Store result in register rt.

Table 4.5 Three-Operand, Register Type-Instruction Summary

Instruction	Format and Description
Add	`ADD rd, rs, rt` Add contents of registers rs and rt and place 32-bit result in register rd. Trap on two's complement overflow.
Add Unsigned	`ADDU rd, rs, rt` Add contents of registers rs and rt and place 32-bit result in register rd. Do not trap on overflow.
Subtract	`SUB rd, rs, rt` Subtract contents of registers rt from rs and place 32-bit result in register rd. Trap on two's complement overflow.
Subtract Unsigned	`SUBU rd, rs, rt` Subtract contents of registers rt from rs and place 32-bit result in register rd. Do not trap on overflow.
Set on Less Than	`SLT rd, rs, rt` Compare contents of register rt to register rs (as signed, 32-bit integers). If register rs is less than rt, rd = 1; otherwise, rd = 0.
Set on Less Than Unsigned	`SLTU rd, rs, rt` Compare contents of register rt to register rs (as unsigned, 32-bit integers). If register rs is less than rt, rd = 1; otherwise, rd = 0.
AND	`AND rd, rs, rt` Bitwise `AND` contents of registers rs and rt and place result in register rd.
OR	`OR rd, rs, rt` Bitwise `OR` contents of registers rs and rt and place result in register rd.
Exclusive OR	`XOR rd, rs, rt` Bitwise exclusive `OR` contents of registers rs and rt and place result in register rd.
NOR	`NOR rd, rs, rt` Bitwise `NOR` contents of registers rs and rt and place result in register rd.

Table 4.6 Shift Instruction Summary

Instruction	Format and Description
Shift Left Logical	`SLL rd, rt, shamt` Shift contents of register rt left by `shamt` bits, inserting zeros into low-order bits. Place 32-bit result in register rd.
Shift Right Logical	`SRL rd, rt, shamt` Shift contents of register rt right by `shamt` bits, inserting zeros into high-order bits. Place 32-bit result in register rd.
Shift Right Arithmetic	`SRA, rd, rt, shamt` Shift contents of register rt right by `shamt` bits, sign-extending the high-order bits. Place 32-bit result in register rd.
Shift Left Logical Variable	`SLLV rd, rt, rs` Shift contents of register rt left. Low-order 5-bits of register rs specify the number of bits to shift. Insert zeros into low-order bits of rt and place 32-bit result in register rd.
Shift Right Logical Variable	`SRLV rd, rt, rs` Shift contents of register rt right. Low-order 5-bits of register rs specify the number of bits to shift. Insert zeros into high-order bits of rt and place 32-bit result in register rd.
Shift Right Arithmetic Variable	`SRAV rd, rt, rs` Shift contents of register rt right. Low-order 5-bits of register rs specify the number of bits to shift. Sign-extend the high-order bits of rt and place 32-bit result in register rd.

Table 4.7 Multiply/Divide Instruction Summary

Instruction	Format and Description
Multiply	`MULT rs, rt` Multiply contents of registers rs and rt as two's complement values. Place 64-bit results in special registers HI and LO.
Multiply Unsigned	`MULTU rs, rt` Multiply contents of registers rs and rt as unsigned values. Place 64-bit results in special registers HI and LO.
Divide	`DIV rs, rt` Divide contents of registers rs and rt as two's complement values. Place 32-bit quotient in special register LO and 32-bit remainder in HI.
Divide Unsigned	`DIVU rs, rt` Divide contents of registers rs and rt as unsigned values. Place 32-bit quotient in special register LO and 32-bit remainder in HI.
Move From HIGH	`MFHI rd` Move contents of special register HI to register rd.
Move From LOW	`MFLO rd` Move contents of special register LO to register rd.
Move To HIGH	`MTHI rd` Move contents of register rd to special register HI.
Move To LOW	`MTLO rd` Move contents of register rd to special register LO.

The execution time of the multiply and divide instructions is shown in Table 4.8.

Table 4.8 Execution Time of Multiply and Divide Instructions

Operation	Clocks
Multiply	17
Divide	17/34

The divide time is shortened to 17 cycles if the divisor has less than 16 significant bits.

In addition to the standard MIPS II instruction set, the CW4011 implements the instruction set extensions described in Table 4.9 to

provide greater application code performance. Table 4.10 and Table 4.11 provide further information on them.

Table 4.9 Instruction Set Extensions

Extension	Description
ADDCIU	Add circular immediate. Does an add immediate, modified according to the value in the new CP0 register CMask. Useful in addressing circular buffers. Important in DSP and other applications that use circular buffers.
FFS/FFC	Find first set/clear. Find first the set/clear bit in the source register, and return the bit number to the destination register. Useful for many applications such as interrupt handlers, floating point emulation, and graphics.
MAX	Maximum. The 32-bit signed contents of two general registers are compared and the greater value is moved to the destination register. This instruction is useful in ABR scheduling algorithms.
MIN	Minimum. The 32-bit signed contents of two general registers are compared and the lesser value is moved to the destination register. This instruction is useful in ABR scheduling algorithms.
RMUL	Rate Multiply. The floating point operands in two general registers are multiplied and the result is stored in LO register. Floating point numbers are expressed in ATM Forum format.
RADD	Rate Add. The floating point operands in two general registers are added and the result is stored in LO register. Floating point numbers are expressed in ATM Forum format.
RSUB	Rate Subtract. The difference of floating point operands is stored in LO register. Floating point numbers are expressed in ATM Forum format.
R2U	Rate to integer conversion. Converts floating point number in ATM Forum format to unsigned 32-bit integer.
SELRR	Select and rotate right. Selects 32 bits from the 64-bit source register pair and rotates the selected data right by the number of bits specified in the new CP0 register ROTATE. Useful for data alignment operation in graphics and in bit-field selection routines for data transmission and compression applications.
U2R	Integer to rate conversion. Converts 32-bit unsigned integer to floating point number in ATM Forum format.

Table 4.9 Instruction Set Extensions (Cont.)

Extension	Description
WAITI	Wait for Interrupt. Halts the CPU in a power saving mode until one of the hardware interrupt lines becomes active. Upon interrupt, normal execution is resumed starting at the interrupt vector address.

Table 4.10 CW4011 ISA Extensions Summary

Instruction	Format and Description
Add Circular Immediate	`ADDCIU rt, rs, immediate` The 16-bit `immediate` is sign extended and added to the contents of general register rs, with the result masked by the value in CP0 register CMask according to the formula: `rt = (rs & !EXP(CMask)) | ((rt + immed) & EXP(CMask))`.
Find First Set	`FFS rt, rs` Starting at the most significant bit in register rs, find the first bit which is set to a one, and return the bit number in register rt. If no bit is set, return with bit 31 of rt set to 1.
Find First Clear	`FFC rt, rs` Starting at the most significant bit in register rs, find the first bit which is cleared, and return the bit number in register rt. If no bit is cleared, return with bit 31 of rt set to 1.
Select and Shift Right	`SELSR rd, rs, rt` Using register rs and rd as a 64-bit register pair, and CP0 register ROTATE as the shift count, shift the register pair rs || rt right the number of bits specified in ROTATE, and place the least significant 32-bit value in result register rd.
Select and Shift Left	`SELSL rd, rs, rt` Using register rs and rd as a 64-bit register pair, and CP0 register ROTATE as the shift count, shift the register pair rs || rt left the number of bits specified in ROTATE, and place the most significant 32-bit value in result register rd.
Minimum	`MIN rd, rs, rt` The two's complement values in registers rs and rt are compared and the smaller value is stored in register rt.
Maximum	`MAX rd, rs, rt` The two's complement values in registers rs and rt are compared and the larger value is stored in register rt.

Table 4.11 APU Rate Instruction Extensions

Instruction	Format and Description
Rate Multiply	`RMUL rs, rt` The 15-bit floating point numbers in registers rs and rt are multiplied and the result is stored in the 15 least significant bits of register LO.
Rate Add	`RADD rs, rt` The 15-bit floating point numbers in registers rs and rt are added and the result is stored in the 15 least significant bits of register LO.
Rate Subtract	`RSUB rs, rt` The 15-bit floating point number in register rt is subtracted from the 15-bit floating point number in register rs. The result is stored in register LO.
Rate to Integer Conversion	`R2U rs` The 15-bit floating point value in register rs is converted to a 32-bit unsigned integer. The result is stored in register LO.
Integer to Rate Conversion	`U2R rs` The 32-bit unsigned integer in register rs is converted into a 15-bit floating point value. The result is stored in register LO.

4.3.4 Jump and Branch Instructions

Jump and branch instructions change the control flow of a program. MIPS I jump and branch instructions always occur with a one-instruction delay. That is, the instruction immediately following the jump or branch is always executed while the target instruction is being fetched from storage. There may be additional cycle penalties, depending on circumstances and implementation, but the penalties are interlocked in hardware. The MIPS II ISA extensions add the branch likely class of instructions that operate exactly like their nonlikely counterparts, except that when the branch is **not** taken, the instruction following the branch is cancelled.

The J-type instruction format is used for both jump and jump-and-link instructions for subroutine calls. In the J-type format, the 26-bit target address is shifted left two bits and combined with the 4 high-order bits of the current program counter to form a 32-bit absolute address.

The R-type instruction format, which takes a 32-bit byte address contained in a register, is used for returns, dispatches, and cross-page jumps.

Branches have 16-bit signed offsets relative to the program counter (I-type). Jump-and-link and branch-and-link instructions save a return address in register 31.

Table 4.12 summarizes the jump instructions, Table 4.13 summarizes the branch instructions, and Table 4.14 summarizes the branch-likely instructions.

Table 4.12 Jump Instruction Summary

Instruction	Format and Description
Jump	`J target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay.
Jump and Link	`JAL target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay. Place address of instruction following delay slot in Link register (R31).
Jump Register	`JR rs` Jump to address contained in register rs with a one-instruction delay.
Jump and Link Register	`JALR rs, rd` Jump to address contained in register rs with a one-instruction delay. Place address of instruction following delay slot in rd.

Table 4.13 Branch Instruction Summary

Instruction	Format and Description
Branch on Equal	`BEQ rs, rt, offset` Branch to target address[1] if register rs is equal to register rt.
Branch on Not Equal	`BNE rs, rt, offset` Branch to target address if register rs does not equal register rt.
Branch on Less than or Equal to Zero	`BLEZ rs, offset` Branch to target address if register rs is less than or equal to 0.
Branch on Greater Than Zero	`BGTZ rs, offset` Branch to target address if register rs is greater than 0.
Branch on Less Than Zero	`BLTZ rs, offset` Branch to target address if register rs is less than 0.
Branch on Greater than or Equal to Zero	`BGEZ rs, offset` Branch to target address if register rs is greater than or equal to 0.
Branch on Less Than Zero And Link	`BLTZAL rs, offset` Place address of instruction following delay slot in Link register (R31). Branch to target address if register rs is less than 0.
Branch on Greater than or Equal to Zero and Link	`BGEZAL rs, offset` Place address of instruction following delay slot in Link register (R31). Branch to target address if register rs is greater than or equal to 0.

1. All branch-instruction target addresses are computed as follows: add address of instruction in delay slot and the 16-bit `offset` (shifted left two bits and sign-extended to 32-bits). All branches occur with a delay of one instruction.

Table 4.14 Branch-Likely Instruction Summary–MIPS II ISA Extensions

Instruction	Format and Description
Branch on Equal Likely	`BEQL rs, rt, offset` Branch to target address[1] if register rs is equal to register rt.
Branch on Not Equal Likely	`BNEL rs, rt, offset` Branch to target address if register rs does not equal register rt.
Branch on Less than or Equal to Zero Likely	`BLEZL rs, offset` Branch to target address if register rs is less than or equal to 0.
Branch on Greater Than Zero Likely	`BGTZL rs, offset` Branch to target address if register rs is greater than 0.
Branch on Less Than Zero Likely	`BLTZL rs, offset` Branch to target address if register rs is less than 0.
Branch on Greater than or Equal to Zero Likely	`BGEZL rs, offset` Branch to target address if register rs is greater than or equal to 0.
Branch on Less Than Zero And Link Likely	`BLTZALL rs, offset` Place address of instruction following delay slot in Link register (R31). Branch to target address if register rs is less than 0.
Branch on Greater than or Equal to Zero and Link Likely	`BGEZALL rs, offset` Place address of instruction following delay slot in Link register (R31). Branch to target address if register rs is greater than or equal to 0.

1. All branch-instruction target addresses are computed as follows: add address of instruction in delay slot and the 16-bit `offset` (shifted left two bits and sign-extended to 32-bits). All branches occur with a delay of one instruction.

4.3.5 Trap Instructions

Trap instructions are part of the MIPS II instruction set and provide instructions that conditionally create an exception, based on the same conditions tested in the branch instructions. (See Table 4.15.)

Table 4.15 Trap Instruction Summary–MIPS II ISA Extensions

Instruction	Format and Description
Trap on Equal	`TEQ rs, rt` Trap if register rs is equal to register rt.
Trap on Equal Immediate	`TEQI rs, immediate` Trap if register rs is equal to the immediate value.
Trap on Greater than or Equal	`TGE rs, rt` Trap if register rs is greater than or equal to register rt.
Trap on Greater Than or Equal Immediate	`TGEI rs, immediate` Trap if register rs is greater than or equal to the immediate value.
Trap on Greater than or Equal Unsigned	`TGEU rs, rt` Trap if register rs is greater than or equal to register rt.
Trap on Greater Than or Equal Immediate Unsigned	`TGEIU rs, immediate` Trap if register rs is greater than or equal to the immediate value.
Trap on Less Than	`TLT rs, rt` Trap if register rs is less than register rt.
Trap on Less Than Immediate	`TLTI rs, immediate` Trap if register rs is less than the immediate value.
Trap on Less Than Unsigned	`TLTU rs, rt` Trap if register rs is less than register rt.
Trap on Less Than Immediate Unsigned	`TLTIU rs, immediate` Trap if register rs is less than the immediate value.
Trap if Not Equal	`TNE rs, rt` Trap if register rs is not equal to rt.
Trap if Not Equal Immediate	`TNEI rs, immediate` Trap if register rs is not equal the immediate value.

4.3.6 Special Instructions

Special instructions cause an unconditional branch to the general exception-handling vector. Special instructions are always R-type and are summarized in Table 4.16.

Table 4.16 Special Instruction Summary

Instruction	Format and Description
System Call	SYSCALL Initiates system call trap, immediately transferring control to exception handler.
Breakpoint	BREAK Initiates breakpoint trap, immediately transferring control to exception handler.

4.3.7 Coprocessor Instructions

The CW4011 supports external (on-chip) coprocessors. In the ATMizer II+ chip, the Coprocessor Condition codes are used to monitor the EDMA Request Queues full status (see Section 4.8.6, “Coprocessor Condition Signals”). Table 4.17 summarizes the coprocessor instructions supported by the APU.

Table 4.17 Coprocessor Instruction Summary

Instruction	Format and Description
Branch on Coprocessor z True (Likely)	`BCzT offset, (BCzTL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32-bits). Branch to the target address (with a delay of one instruction) if coprocessor z’s condition line is true. In the case of Branch Likely, the delay slot instruction is not executed when the branch is not taken.
Branch on Coprocessor z False (Likely)	`BCzF offset, (BCzFL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32-bits). Branch to the target address (with a delay of one instruction) if coprocessor z’s condition line is false. In the case of Branch Likely, the delay slot instruction is not executed when the branch is not taken.

4.3.8 System Control Coprocessor (CP0) Instructions

Coprocessor 0 instructions perform operations on the system control coprocessor (CP0) registers to manipulate the exception-handling facilities of the processor. Table 4.18 summarizes the CP0 instructions and Table 4.19 shows the extension.

Table 4.18 CP0 Instruction Summary

Instruction	Format and Description
Move To CP0	`MTC0 rt, rd` Load contents of CPU register rt into CP0 register rd. This instruction has one delay slot and should be followed by one no operation (NOP) instruction.
Move From CP0	`MFC0 rt, rd` Load contents of CP0 register rd into CPU register rt. This instruction has one delay slot and should be followed by one no operation (NOP) instruction.
Exception Return	`ERET` Load the PC from ErrorEPC (SR2 = 1:Error Exception) or EPC (SR2 = 0: Exception) and clear ERL bit (SR2 = 1) or EXL bit (SR2 = 0) in the Status register. SR2 is Status register bit[2].
Restore From Exception	`RFE` Restore previous interrupt mask and mode bits of the Status register into current status bits. Restore old status bits into previous status bits.

Table 4.19 CP0 Instruction Extension Summary

Instruction	Format and Description
Wait for Interrupt	`WAITI` Stops execution of instructions and places the processor into a power save condition until a hardware interrupt or reset is received. This instruction must be followed by two or more no operation (NOP) instructions.

4.3.9 Cache Maintenance Instructions

Cache Maintenance instructions are always I-type. Cache instructions must be followed by three no operation (NOP) instructions. Table 4.20 summarizes these instructions.

Table 4.20 Cache Maintenance Instruction Summary

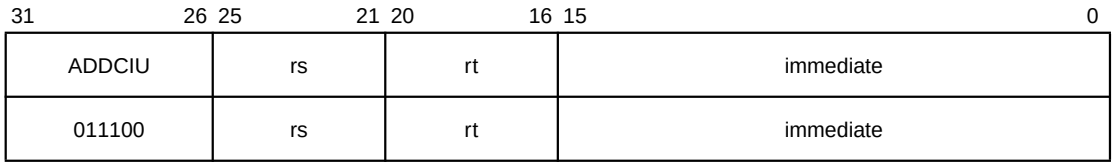
Instruction	Format and Description
Flush I-Cache	`FLUSHI` Flush I-Cache. 256 stall cycles will be needed.
Flush D-Cache	`FLUSHD` Flush D-Cache. 256 stall cycles will be needed.
Flush I-Cache & D-Cache	`FLUSHID` Flush both I-Cache and D-Cache in 256 stall cycles.
WriteBack	`WB offset(base)` Write back a D-Cache line addressed by `offset` + `GPR[base]`.

4.3.10 APU and CW4011 Instruction Set Extensions

The new instruction set extensions, including the cache maintenance instructions, are further defined in this section. They are listed in alphabetical order, one instruction to a page.

ADDCIU **Add with Circular Mask Immediate**

Format



31 — 26	25 — 21	20 — 16	15 — 0
ADDCIU	rs	rt	immediate
011100	rs	rt	immediate

Syntax `ADDCIU rt, rs, immediate`

Description The `immediate` field of the instruction is sign-extended and added to the contents of general register rs, the result of which is masked with the expanded value in special register CMask according to the equation. The CMask register is CP0 register number 24. Its valid bits are [4:0].

T: $sign\_extend\_immed = (immediate_{15})^{16} \,||\, immediate_{15..0}$
$GPR[rt] = GPR[rs]_{31..cmask} \,||\, (GPR[rs] + sign\_extend\_immed)_{cmask-1..0}$

Exceptions None

FFC Find First Clear Bit

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	rd	0	FFC
000000	rs	0	rd	00000	001011

Syntax

```
FFC rd, rs
```

Description

The contents of general register rs are examined starting with the most significant bit. The bit number of the first clear bit is returned in general register rd. If no bit is set, all ones are returned in rd.

Exceptions

None

FFS Find First Set Bit

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	rd	0	FFS
000000	rs	0	rd	00000	001010

Syntax

FFS `rd, rs`

Description

The contents of general register rs are examined starting with the most significant bit. The bit number of the first set bit is returned in general register rd. If no bit is set, all ones are returned in rd.

Exceptions

None

FLUSHD FLUSH Data Cache

Format

31 26	25 21	20 16	15 0
CACHE	0	FLUSHD	0
101111	00000	00010	0

Syntax

FLUSHD

Description

FLUSHD flushes all Data Cache lines and causes 256 clocks of stall cycles regardless of the cache size. This instruction must be followed by three no operation (NOP) instructions.

Note: Cache instructions only work on enabled cache sets. See the IE0, IE1, DE0, and DE1 bits in the CCC register, page 4-74.

Exceptions

None

FLUSHI FLUSH Instruction Cache

Format

31 26	25 21	20 16	15 0
CACHE	0	FLUSHI	0
101111	00000	00001	0

Syntax

`FLUSHI`

Description

`FLUSHI` flushes all Instruction Cache lines and causes 256 clocks of stall cycles regardless of the cache size. This instruction must be followed by three no operation (`NOP`) instructions.

Note: Cache instructions only work on enabled cache sets. See the `IE0`, `IE1`, `DE0`, and `DE1` bits in the CCC register, page 4-74.

Exceptions

None

FLUSHID FLUSH Instruction and Data Cache

Format

31 26	25 21	20 16	15 0
CACHE	0	FLUSHID	0
101111	00000	00011	0

Syntax

`FLUSHID`

Description

`FLUSHID` flushes all Data and Instruction Cache lines and causes 256 clocks of stall cycles regardless of the cache size. This instruction must be followed by three no operation (`NOP`) instructions.

Note: Cache instructions only work on enabled cache sets. See the `IE0`, `IE1`, `DE0`, and `DE1` bits in the CCC register, page 4-74.

Exceptions

None

SELSL Select and Shift Left

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SELSL
000000	rs	rt	rd	00000	000101

Syntax

SELSL rd, rs, rt

Description

The contents of general register rs and the contents of general register rt are combined to form a 64-bit doubleword. The doubleword is shifted left the number of bits specified in CP0 register ROTATE, and the upper 32-bits of the result are placed in general register rd. This ROTATE register is CP0 register number 23, with valid bits [4:0].

```
T:      s <- ROTATE4..0
        GPR[rd] <- GPR[rs]31-s..0 || GPR[rt]31..32-s
```

Exceptions

None

SELSR **Select and Shift Right**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	SELSR
000000	rs	rt	rd	00000	000001

Syntax `SELSR rd, rs, rt`

Description The contents of general register rs and the contents of general register rt are combined to form a 64-bit doubleword. The doubleword is shifted right the number of bits specified in CP0 register ROTATE, and the lower 32-bits of the result are placed in general register rd. This ROTATE register is CP0 register number 23. Valid bits are [4:0].

```
T:      s <- ROTATE4..0
        GPR[rd] <- GPR[rs]s-1..0 || GPR[rt]31..s
```

Exceptions None

WAITI — Wait for Interrupt

Format

31 26	25 21	20 16	15 11	10 6	5 0
COP0		0	0	0	WAITI
000000	10000	00000	00000	00000	100000

Syntax

```
WAITI
```

Description

Execution of this instruction causes the main processor clock to stop and halts all instruction execution. Execution resumes with reception of a hardware interrupt, NMI, or reset exception. While in wait mode, the processor is in a power saving mode, using very little current because the clock is turned off to most of the circuitry.

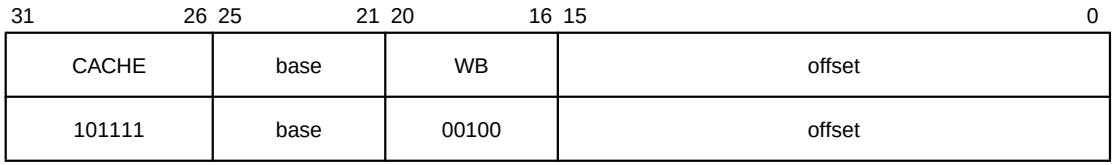
`WAITI` must be followed by two or more no operation (`NOP`) instructions; otherwise, the results are undefined.

Exceptions

None

WB **WriteBack Data Cache**

Format



31 26	25 21	20 16	15 0
CACHE	base	WB	offset
101111	base	00100	offset

Syntax `WB offset (base)`

Description Eight words of the Data cache line addressed by `offset + GPR[base]` are written back to memory if the line is dirty. Upper bits of `offset + GPR[base]` are ignored. This instruction must be followed by three no operation (`NOP`) instructions.

Exceptions None

4.3.11 ATMizer II+ Instruction Set Extensions

This section defines the instruction set extensions that support available bit rate (ABR) calculations.

MAX **Maximum**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	MAX
000000	rs	rt	rd	00000	101001

Syntax `MAX rd, rs, rt`

Description The contents of general register rs and the contents of general register rt are compared with both operands treated as 32-bit signed values. The maximum value is stored in general register rd.

Note: This instruction may be used to identify the maximum of rate floating point numbers.

Exceptions None

Min **Minimum**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	rd	0	MIN
000000	rs	rt	rd	00000	101000

Syntax `MIN rd, rs, rt`

Description The contents of general register rs and the contents of general register rt are compared with both operands treated as 32-bit signed values. The minimum value is stored in general register rd.

Note: This instruction may be used to identify the minimum of rate floating point numbers.

Exceptions None

RADD **Rate Addition**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	RADD	MULT
000000	rs	rt	0	00010	011000

Syntax `RADD rs, rt`

Description The 15 least significant bits of general register rs and the 15 least significant bits of general register rt are added with both operands treated as 15-bit floating point numbers. The result is stored in special register LO. The 17 most significant bits of register LO are clear.

Exceptions `IntRateExc` exception is generated on floating point overflow or underflow.

RMUL **Rate Multiply**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	RMUL	MULT
000000	rs	rt	0	00001	011000

Syntax `RMUL rs, rt`

Description The 15 least significant bits of general register rs and the 15 least significant bits of general register rt are multiplied with both operands treated as 15-bit floating point numbers. The result is stored in special register LO. The 17 most significant bits of register LO are clear.

Exceptions `IntRateExc` exception is generated on floating point overflow or underflow.

RSUB Rate Subtraction

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	RSUB	MULT
000000	rs	rt	0	00011	011000

Syntax

`RSUB rs, rt`

Description

The 15 least significant bits of general register rt are subtracted from the 15 least significant bits of general register rs with both operands treated as 15-bit floating point numbers. The result is stored in special register LO. The 17 most significant bits of register LO are clear.

Exceptions

`IntRateExc` exception is generated on floating point overflow or underflow.

R2U — Rate to Integer Conversion

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	0	R2U	MULT
000000	rs	0	0	00100	011000

Syntax

R2U `rs`

Description

The 15 least significant bits of general register rs are converted from the rate floating point format to an unsigned 32-bit integer. The result is placed in special register LO. The fractional portion of the floating point number is truncated.

Exceptions

None

U2R Integer to Rate Conversion

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	0	0	U2R	MULT
000000	rs	0	0	00101	011000

Syntax

`U2R rs`

Description

The 32-bit unsigned integer in general register rs is converted into the rate floating point format. The result is stored in the 15 least significant bits of special register LO. The 17 most significant bits of register LO are clear.

Exceptions

None

4.4 CP0 Data Manipulation Registers

The System Control Coprocessor (CP0) contains two registers that are used by the CW4011 instruction set extensions for data manipulation operations. They are the:

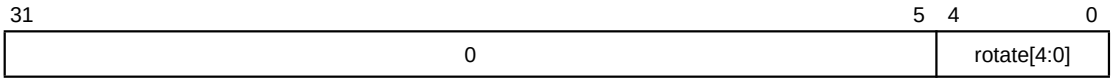
- Rotate Register (CW4011 register 23)
- CMask Register (CW4011 register 24)

The CP0 also includes registers that are used for exception handling. For a description of these CP0 registers, see Section 4.6.2, “Exception Handling Registers.”

4.4.1 Rotate Register (23)

Figure 4.6 shows the format of the Rotate register. The CW4011 instruction set extensions use the Rotate register. Specifically, the Select and Shift Right (SELSR) instruction and the Select and Shift Left (SELSL) instruction use the five-bit rotate value as the shift count. These instructions are used for graphic’s data alignment operations and bit-field selection routines required for data transmission and compression applications.

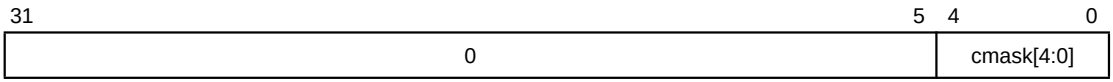
Figure 4.6 Rotate Register



4.4.2 Circular Mask Register (24)

Figure 4.7 shows the format of the CMask register. The CW4011 instruction set extensions use the Circular Mask (CMask) register. The Load/Store (word/halfword/byte) with update circular instructions store a value in the destination register and update the base address register with the addition of base + offset, modified according to the five-bit value contained in the CMask register. This capability is useful in DSP and other applications that use circular buffers.

Figure 4.7 CMask Register



4.5 Cache Memory

The APU instruction cache and data cache include the following features:

- Direct mapped or two-way set associative is selectable for I-Cache and D-Cache, respectively. A Least Recently Used (LRU) algorithm is used for two-way set associative cache replacement.
- The I-Cache consists of two sets of 4 Kbytes (8 Kbytes total). The D-Cache consists of two sets of 2 Kbytes (4 Kbytes total). Each set is configurable as either direct-mapped or two-way set associative.
- For load and store instructions, the cache performs a tag check. The physical address of the data is compared with the cache tag to determine if there is a cache hit or miss.
- One cache line is 8 words (4 doublewords = 32 bytes = 256 bits). Refill address ordering is wraparound from the missing address.
- Cache Write Back or Write Through operation can be selected with the `WB` bit in the CCC register.
- The Cache RAM can be configured for use as a Scratch-Pad RAM.

4.5.1 Cache States

The I-Cache has two states. The D-Cache states depend on whether the D-Cache is in Write-Through or Write-Back mode.

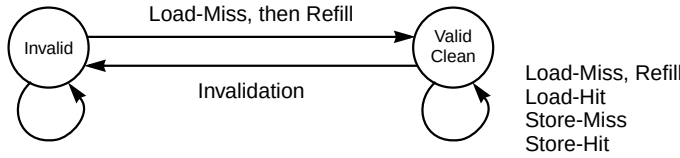
4.5.1.1 I-Cache and WriteThrough D-Cache States

The I-Cache and Write-Through mode of the D-Cache have two states, Invalid and Valid Clean. Each cache line has a Validity (V) bit that indicates whether the line is Invalid (V bit = 0) or Valid Clean (V bit = 1). Initialization sets all cache lines to Invalid. This invalidation can be done using a Cache Flush instruction (see page 4-55) or by setting the Invalidate mode bit in the CCC register (see page 4-56).

Figure 4.8 shows the state diagram for the I-Cache and Write Through D-Cache. The state of a cache line changes from Invalid to Valid Clean when the line is refilled after a cache miss occurs. The cache state remains Valid Clean until the line is invalidated by another Cache Flush instruction (see page 4-55) or if the invalidate mode bit is set in the

CCC register (see page 4-56 and Section 4.6.2.9, “Configuration and Cache Control Register (16)”).

Figure 4.8 I-Cache and D-Cache State Diagram



4.5.1.2 Write Back D-Cache States

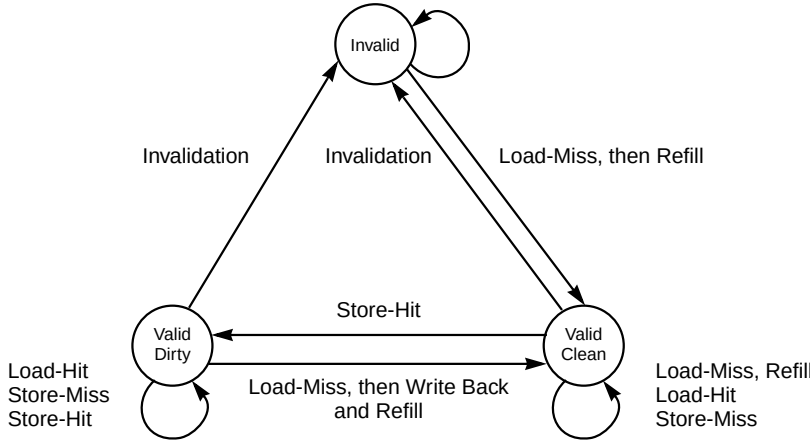
When the D-Cache operates in WriteBack mode, there are three states for each cache line; Invalid, Valid Clean, and Valid Dirty. The V bit and WB bit of each line indicate the state as shown in Table 4.21.

Table 4.21 D-Cache Write-Back Mode

State	V Bit	WB Bit	Condition
Invalid	0	X	The cache line does not contain valid information.
Valid Clean	1	0	The cache line includes valid information consistent with memory.
Valid Dirty	1	1	The cache line includes valid information, but it is not consistent with memory.

Figure 4.9 shows a state diagram of the D-Cache in Write-Back mode.

Figure 4.9 D-Cache Write Back State Diagram



A store-hit occurs when the Validity (V) bit is set and the Tag address in the Tag RAM matches the physical address of the store data.

When a store-miss occurs, the store data is not written into the D-Cache, and the state condition of the cache line is not changed. Instead, the store data is written to the external Write Buffer. From the Write Buffer the data is written to external main memory.

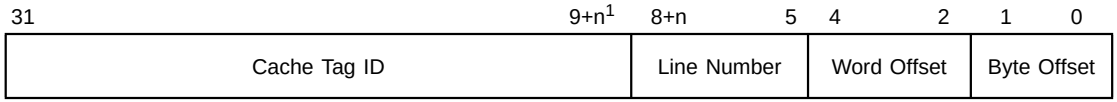
In the Write-Back mode, some lines, known as dirty lines, contain more recent information than the main memory. Dirty line data can be written back to main memory using the Write-Back Data Cache (WB) instruction (refer to page 4-41).

One Write-Back Data Cache instruction can write back each line of both sets in a two-way set associative configuration. The instruction does not check the address for hit or miss. If the Write Back (WB) state bit for a cache line is set, the line data is written back to main memory. Write back requires several stall cycles to read the data from the D-Cache.

4.5.2 Address and Cache Tags

Figure 4.10 illustrates the format of an instruction or data address for both direct and two-way set associative caches. The Cache Tag ID field contains the physical tag address for this line of cache data. The Line Number field provides the address of the cache line, which is an eight-word block of cache data. The Word Offset field addresses one word in the cache line.

Figure 4.10 Cache Address Format



1. n = 2 for D-Cache and 3 for I-Cache

4.5.3 D-Cache Scratch-Pad RAM Mode

Either Data Cache Set 0 or Set 1 RAM can be configured as a local, high-speed, Scratch-Pad RAM. An access to the Scratch-Pad RAM is a local memory access without stall cycles. The Scratch-Pad RAM must be located in a cacheable virtual address space. See Section 4.7.1, "Operating Modes." The desired address of the Scratch-Pad RAM must be programmed into the cache Tag RAM (all locations) before enabling Scratch-Pad RAM mode using the SR bits in the CCC register (see Section 4.6.2.9, "Configuration and Cache Control Register (16)").

To program a Data Tag RAM enter cache maintenance mode and isolate the desired Tag RAM: CCC register IsC bit = 1, Inv bit = 0, Tag bit = 1 and desired DE1/DE0 bit = 1. The CCC register is accessed using the mfc0 and mtc0 instructions (see Section 4.5.5.3, "Cache Maintenance"). Once cache maintenance mode is entered, all load and store instructions access the selected Tag RAM using the format shown in Figure 4.11. Each line of the Tag RAM should be programmed with the same value.

Tag Data = location of the Scratch-Pad RAM (Address bits[31: 9+n])

HT = 0

V = 1 valid bit must be set

WB = 0

Tag locations are accessed on eight-word boundaries at the following addresses:

0x0000 = tag location 0

0x0020 = tag location 1

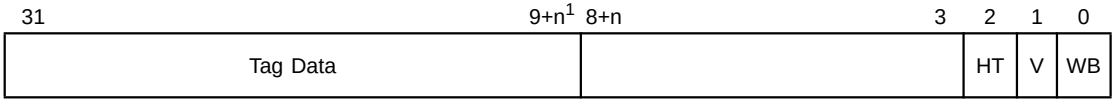
0x0040 = tag location 2

0x0060 = tag location 3

0x0080 = tag location 4

Once the Tag RAM has been programmed, Scratch-Pad RAM is enabled by setting the corresponding DE bit = 0 and SR bit = 1 in the CCC register.

Figure 4.11 Tag RAM Access Format



1. n = 2 for D-Cache and 3 for I-Cache

If the D-Cache Scratch-Pad RAM is enabled, an access to the Scratch-Pad RAM area is a local memory access without any stall cycle.

4.5.4 I-Cache RAM Mode

Set 1 of the APU I-Cache can be configured as an Instruction RAM by setting the IR bit (26) in the CCC register (page 4-74) to one. This allows you to lock a set of frequently used code into the Instruction Cache.

You cannot configure Instruction Cache Set 0 as Instruction RAM. However, you can achieve the same effect by leaving both sets as true caches, mapping code that should reside in an Instruction RAM into a cacheable memory area, and mapping all other code into an uncacheable area. For this to work, all cacheable code must be contained within two 4 Kbyte blocks.

To configure Instruction Cache Set 1 as Instruction RAM, first program its Tag RAM with the desired address and set the valid bit. The Tag RAM is accessed in cache maintenance mode: CCC register IsC bit = 1, Inv bit = 0, Tag bit = 1, and IE1 bit = 1. Once cache maintenance mode is entered, all load and store instructions access the selected Tag RAM using the format shown in Figure 4.11. Each line of the Tag RAM should be programmed with the same value.

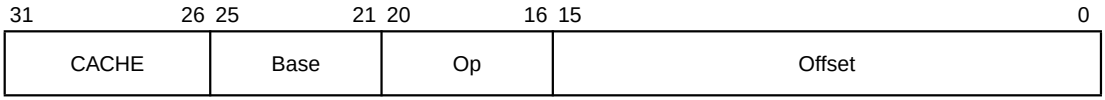
Then download the desired code to Instruction Cache Set 1 Data RAM. The cache RAM is accessed in cache maintenance mode: CCC register IsC bit = 1, Inv bit = 0, Tag bit = 0, and IE1 bit = 1. Once cache maintenance mode is entered all load and store instructions access the Set 1 Data RAM. You will need to toggle in and out of cache maintenance mode to perform the download.

Once the Tag RAM and Data RAM have been programmed, Instruction RAM mode is entered by setting CCC register bits IE1 = 1 and IR1 = 1.

4.5.5 Cache Instructions

Figure 4.12 shows the cache instruction format.

Figure 4.12 Cache Instruction Format



CACHE — **Cache Instruction** **[31:26]**
Specifies this is a Cache instruction type. The bit coding is 0b101111.

Base — **Base Register** **[25:21]**
Specifies the base address register to be used in the effective address calculation.

Op — **Cache Operation Selection Code** **[20:16]**
Bits [20:18] specify either cache flush or D-Cache Write Back. For cache flush operation, bits [17,16] specify which cache to flush (instruction, data, or both). For write-back operation, bits [17,16] are ignored.

A cache flush operation invalidates all cache lines in either the instruction or data cache or both. Bit encoding is defined as follows:

Op[4:0]	Definition
0b00001	Flush instruction cache (CACHE_FLUSHI)
0b00010	Flush data cache (CACHE_FLUSHD)
0b00011	Flush instruction and data cache (CACHE_FLUSHID)
0b001XX	Cache Write Back to data cache only

Offset — **Offset Value** **[15:0]**
The offset value is added to the contents of the base register to generate the effective address.

4.5.5.1 Flush (All Cache Invalidation)

Execution of a cache instruction can invalidate all lines of the D-Cache and/or I-Cache. Bits 16 and 17 of the instruction specify whether the instruction is effective for the D-Cache, the I-Cache, or both caches. If both bits 16 and 17 are zero, the instruction performs a No Operation

(NOP). The base register and the offset fields have no meaning in this instruction.

Note: Cache instructions only work on enabled cache sets. See the IE0, IE1, DE0, and DE1 bits in the CCC register, page 4-74.

The instruction invalidates one cache line in one or more cache sets in one clock cycle. The instruction starts in the WB pipeline stage, and the pipeline stall request signal is asserted while the cache lines are being invalidated. If the pipeline cancel signal is asserted, the invalidation is not executed. The number of the invalidation cycles is always 256.

4.5.5.2 Write Back

Write Back is effective for the D-Cache only, so bits 16 and 17 of the instruction are ignored. Bits[12:5] of the effective address, which is offset + GPR[base], specify the D-Cache line. One cache instruction writes back both lines of the two-way set associative cache if the WB bit is set in the Tag RAM.

If the WB bit is zero, a NOP is performed. Write Back is executed in the WB pipeline stage and causes four stall cycles to read data from a dirty line. The WB bits are cleared after the cache lines are written back.

4.5.5.3 Cache Maintenance

The CCC register cache control bits, shown in Table 4.22, can be used for D-Cache and I-Cache maintenance and testing. (See page 4-74 for a complete description of the CCC register.)

Table 4.22 Cache Control Bits

Bit(s)	Function
IR1[1]	Instruction RAM Enable (1)/Disable (0)
IE0	I-Cache Set 0 Enable (1)/Disable (0)
IE1[1]	I-Cache Set 1 Enable (1)/Disable (0)
IS[1:0]	I-Cache Size (00, 1 Kbyte; 01, 2 Kbytes; 10, 4 Kbytes; 11, 8 Kbytes)
DE0[2]	D-Cache Set 0 Disable (1)/Enable (0)
DE1[2]	D-Cache Set 1 Disable (1)/Enable (0)

Table 4.22 Cache Control Bits

Bit(s)	Function
DS[1:0]	D-Cache Size (00, 1 Kbyte; 01, 2 Kbytes; 10, 4 Kbytes; 11, 8 Kbytes)
WB	D-Cache Write Back (1)/Write Through.....if TE bit = 1
SR0[2]	D-Cache Set 0 Scratch-Pad RAM Enable (1)/Disable (0)
SR1[2]	D-Cache Set 1 Scratch-Pad RAM Enable (1)/Disable (0)
IsC	D-Cache/I-Cache Isolate Cache Mode Enable (1)/Disable (0)
TAG	D-Cache/I-Cache Tag Test Mode Enable (1)/Disable (0)
INV	D-Cache/I-Cache Invalidate Mode Enable (1)/Disable (0)

1. IRAM mode: IR1 = 1, IE1 = 1
2. SRAM mode: DE0 = 0. DE1 = 0, SR0 = 1, SR1 = 1

Maintenance Modes – The APU has three maintenance modes that allow you to maintain and test the internal I-Cache and D-Cache. The three modes are:

- Data Test
- Tag Test
- Invalidate

Preparation – Before entering any of the three modes, the processor must be executing in kseg1 (noncacheable address space), then interrupts must be disabled and the caches must be isolated (IsC bit = 0). When the caches are isolated, load and store instructions access the I-Cache and D-Cache.

To enable the cache maintenance mode, use the following procedure:

Step 1. Set up the CCC register.

Set the appropriate bits in the CCC register with IsC bit = 1. This step is performed by executing an MTC0 instruction. The MTC0 instruction has one delay slot. The instruction immediately following the MTC0 instruction should not be a load or store.

The IS0, IS1, DS0, and DS1 bits select the cache to access. Only one cache should be enabled when performing loads. Multiple caches may be enabled when performing stores. The TAG and INV bits select the cache maintenance mode. Table 4.23 lists the encoding for the two bits.

Table 4.23 TAG and INV Encoding

TAG	INV	Cache Maintenance Mode
0	0	Data Test
1	0	Tag Test
x	1	Invalidate

Step 2. Disable Interrupts.

Step 3. Clear the IE bit in the Status register (page 4-65, page 4-68) to disable all interrupts. This operation is usually done automatically because cache maintenance operations are done in an exception handler (most commonly the reset handler). For information about the Status register, refer to page 4-65 and page 4-68.

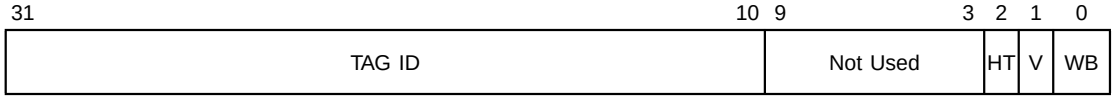
Data Test Mode – In this mode, all loads and stores access the data RAMs selected by the IE0, IE1, DE0, and DE1 bits. The effective lower address bits of the load or store instruction specify the cache address.

Tag Test Mode – When the TAG bit is set to one, the APU is in Tag Test mode. Load and store operations access the Tag RAMs. The tag bits available for testing in the Tag Test mode are the Tag ID bits, Hit (HT) bit, Validity (V) bit, and the Write Back (WB) bit. Note that the WB bit is present only in D-Cache. The HT bit is ignored during a store operation. For a load operation, the HT bit is set if a match occurs.

The Tag ID bits are written from or compared to the most significant bits of the effective address (offset + GPR[base]).

A Tag RAM load operation returns the information shown in Figure 4.13.

Figure 4.13 Tag Test Mode Format



Invalidate Mode – When the INV bit is set to one, the APU is in Invalidate mode. Because the caches contain random data on both warm and cold starts, software must invalidate all lines in the I-Cache and D-Cache. Executing store word instructions invalidates the addressed cache line in the enabled cache(s). After reset, zeroes must be written into all Tags for both sets of D-Cache and I-Cache.

4.6 Exceptions

When the CW4011 detects an exception, it suspends the normal sequence of instruction execution. The processor then disables interrupts and forces execution of a software handler located at a fixed address in memory. (The External Vectored Interrupt Exception uses a separate routine, the EVI handler, as described in Section 4.8.4, "Vectored Interrupt Processing.") The handler saves the context of the processor and restores it after the handler services the exception condition.

When an exception occurs, CP0 loads the Exception Program Counter (EPC) register (page 4-73) with a restart location where program execution can resume after the exception has been serviced.

The restart location in the EPC register is the address of the instruction that caused the exception or, if the instruction was executing in a branch delay slot, it is the address of the branch instruction immediately preceding the delay slot. The instruction causing the exception and all the instructions following it in the pipeline are aborted.

All events that can initiate exception processing are described in this section and are listed in Table 4.24 below.

Table 4.24 APU Exceptions

Exception	Cause
Cold Reset	Deassertion of the CW4011 “cold reset” signal (PCI_RSTn pin or XPP_APU_Reset bit).
Warm Reset	Assertion and deassertion of the WRESETn signal.
NMI	Assertion of the CW4011 NMI signal.
Address Error	Attempt to load, fetch, or store an unaligned word, that is, a word or halfword at an address not evenly divisible by four or two, respectively. Also caused by references to an address with the most significant bit set while in user mode.
TBL Refill	No TBL entry to match a reference to a mapped address space.
TBL Invalid	Virtual address reference matches a TBL entry that is marked invalid.
TBL Modified	Virtual address reference to memory during a store operation matches a TBL entry that is marked valid but is not dirty or writable.
Bus Error	Assertion of the CW4011 external “bus error” signal. (Refer to Watchdog Timer on page 4-112 and Bus Error on page 4-88)
Integer Overflow	Two’s complement overflow during an add or subtract.
Trap	A Trap instruction results in a “true” condition.
System Call	An attempt to execute the SYSCALL instruction.
Breakpoint	An attempt to execute the BREAK instruction.
Reserved Instruction	Execution of an instruction with an undefined or reserved major operation code (bits [31:26]) or a SPECIAL instruction whose minor opcode (bits [5:0]) is undefined.
Interrupt	Assertion of one of CW4011’s six hardware interrupt inputs, or by setting one of the two software interrupt bits in the Cause register (page 4-71). Interrupts must be enabled.
Ext Vectored Interrupt	Assertion of an external vectored interrupt (refer to page 4-93).
Debug	Detection of a program counter breakpoint, data address breakpoint, or trace.

The remaining sections of this chapter provide more detail on the organization and operation of the CW4011 exception handling. The three major topic areas are:

- R3000 Exception Compatibility Mode
- Exception Handling Registers
- CW4011 Exceptions

4.6.1 R3000 Exception Compatibility Mode

Although the CW4011 processor is based on the MIPS R4000 architecture, an R3000 style exception processing capability has been added. This allows you to configure CP0 exception processing so existing R3000 exception handling code runs on the CW4011 processor with little or no modification.

This R3000 compatibility mode is under control of the compatibility bit (bit 24) of the Configuration and Cache Control (CCC) register (page 4-74). The compatibility bit is cleared (R4000 mode) upon a cold reset exception. If R3000 mode operation is desired, bit 24 should be set as part of the cold reset handler. Once placed in R3000 mode, the processor should only be switched back to R4000 mode through the initiation of another cold reset. When R3000 mode is enabled, the behavior of the following areas is affected:

- Status Register
- Exception Handling Vectors
- Exception Return (RFE vs. ERET)

4.6.1.1 Status Register

The lower six bits of the Status register (page 4-65) are redefined to implement the kernel/user mode and interrupt enable stack as defined by the R3000 architecture.

4.6.1.2 Exception Handling Vectors

The exception handling vectors (base and offset) are remapped to those specified by the R3000 architecture. The Exception Vectors are discussed in detail later in this chapter.

4.6.1.3 Exception Return

When operating in R3000 compatibility mode, exception return is accomplished using the RFE instruction. If an attempt is made to use the ERET instruction, a Reserved Instruction exception will be recognized.

In the remainder of this chapter, the differences between standard operation (R4000) and R3000 compatibility mode are indicated where appropriate. In all other cases, operation for the two modes is identical.

4.6.2 Exception Handling Registers

This section describes the CP0 registers that are used in exception processing. Software examines these registers, shown in Table 4.25, during exception processing to determine the cause of an exception and the state of the CPU at the time of the exception. Each of these registers is described in detail in the sections that follow.

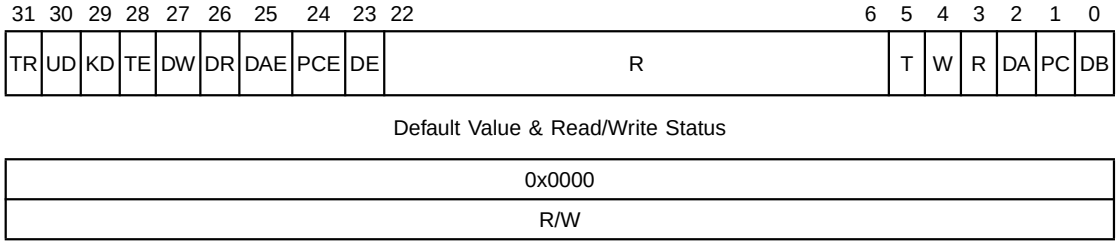
Table 4.25 CP0 Exception Processing Registers

Register Name	CP0 Register Number
DCS (Debug Control and Status)	7
Count	9
Compare	11
Status	12
Cause	13
EPC (Exception Program Counter)	14
PRId (Processor Revision Identifier)	15
CCC (Configuration & Cache Control)	16
LLAdr (Load Linked Address)	17
BPC (Breakpoint Program Counter)	18
BDA (Breakpoint Data Address)	19
BPCM (Breakpoint PC Mask)	20
BDAM (Breakpoint Data Address Mask)	21
Error EPC	30

4.6.2.1 Debug Control and Status (DCS) Register (7)

The Debug Control and Status (DCS) register contains the enable and status bits for the CW4011 debug facility. All bits have read/write access. Figure 4.14 shows the format of the DCS register.

Figure 4.14 DCS Register



TR **Trap Enable** **31**
When set, enables debug exception vector 0. When cleared, cannot trap, but updates the debug status bits with the debug event information.

UD **User Mode Debug Enable** **30**
Not used in L64364.

KD **Kernel Mode Debug Enable** **29**
When set, enables debug event detection in Kernel mode.

TE **Trace Event Enable** **28**
When set, enables trace event detection. A trace event is a nonsequential fetch.

DW **Data Write Enable** **27**
When set, enables data write at Breakpoint Data Address (BDA) event detection (refer to page 4-78). The DAE bit must also be set.

DR **Data Read Enable** **26**
When set, enables data read at Breakpoint Data Address event detection. The DAE bit must also be set.

DAE **Data Address Enable** **25**
When set, enables data address breakpoint debug events.

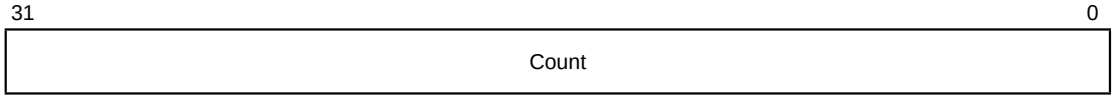
PCE	**Program Counter Enable** When set, enables Program Counter breakpoint debug events.	**24**
DE	**Debug Enable** When set, enables debugging. When cleared, disables debugging.	**23**
R	**Reserved** These bits should all be cleared to 0.	**[22:6]**
T	**Trace Event Detected** When set, indicates a trace event was detected.	**5**
W	**Write Reference Match** When set, indicates a write reference-matching Breakpoint Data Address was detected.	**4**
R	**Read Reference Match** When set, indicates a read reference-matching Breakpoint Data Address was detected.	**3**
DA	**Data Address Debug Detected** When set, indicates a data address debug was detected.	**2**
PC	**Program Counter Debug Detected** When set, indicates a Program Counter debug condition was detected.	**1**
DB	**Debug Condition Detected** When set, indicates a debug condition was detected.	**0**

4.6.2.2 Count Register (9)

The Count register acts as a timer, incrementing at a constant rate. It increments regardless of whether an instruction is executed, retried, or any forward progress is made. The Count register increments at half the maximum instruction issue rate.

The Count register is a read/write register that can be written to for diagnostic purposes. Figure 4.15 shows its format.

Figure 4.15 Count Register

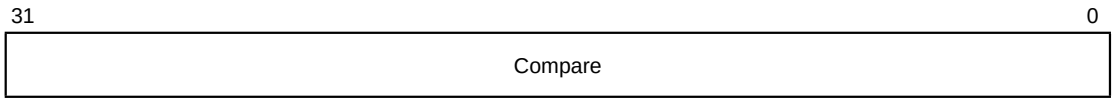


4.6.2.3 Compare Register (11)

The Compare register, shown in Figure 4.16, can be set to generate a timed interrupt. When the timer facility is enabled by the TMR bit (19) in the CCC register (page 4-74) and the value of the Count register (page 4-64) reaches the value written in the Compare register, interrupt IP7 (bit 15) in the Cause register (page 4-71) is set, causing an interrupt on the next execution cycle if the interrupt is enabled. Writing a value to the Compare register clears the timer interrupt.

The Compare register is read/write in diagnostic mode and write-only in normal operation.

Figure 4.16 Compare Register



4.6.2.4 Status Register (12) - R4000 Mode

The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. There are two versions of the Status register; one for the R4000 compatibility mode of the CW4011 and one for the R3000 compatibility mode. The compatibility mode is determined by the CMP bit (24) of the CCC register (page 4-74).

The format of the R4000 version of the Status register is shown in Figure 4.17.

Figure 4.17 Status Register (R4000 Mode)

31–28	27–23	22	21	20	19–16	15–10	9–8	7–6	5	4–3	2	1	0
CU	R	BEV	R	SR	R	INT	SW	R	R	KSU	ERL	EXL	IE
Default Value & Read/Write Status													
0xF	0x00	1	0	0	0x0	XXX[1]	X	0x0	X	XX	1	X	X
R/W													

1. X = undefined at reset.

CU[3:0]	**Coprocessor Enables** Each bit controls a coprocessor enable/disable function. For the L64364, all four bits must be set, enabling all four coprocessors.	**[31:28]**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**[27:23]**
BEV	**Exception Vector** Controls the location of general exception vectors. When cleared, the location is normal (0x8000.0000). When set, the location is the bootstrap (0xBFC0.0000). See Section 4.6.3.3, “Exception Vector Locations.”	**22**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**21**
SR	**Soft Reset** When set, this bit indicates that a soft reset has occurred.	**20**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**[19:16]**
INT[5:0]	**Hardware Interrupt Mask** When set, enables the corresponding hardware interrupt. When cleared, disables the hardware interrupt.	**[15:10]**
SW[1:0]	**Software Interrupt Mask** When set, enables the corresponding software interrupt. When cleared, disables the software interrupt.	**[9:8]**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**[7:6]**
R	**Reserved** Not used in the L64364.	**5**
KSU[1:0]	**Base Mode** These two bits are encoded to specify either kernel mode (0b00) or user mode (0b10). All other bit combinations are reserved.	**[4:3]**

ERL	**Error Level** When set, the error level is enabled. When cleared, the error level is disabled.	**2**
EXL	**Exception Level** When set, the exception level is enabled. When cleared, the exception level is disabled.	**1**
IE	**Interrupt Enable** When set, interrupts are enabled. When cleared, interrupts are disabled.	**0**

Interrupt Enable – Interrupts are enabled when all of the following conditions are true:

- IE (Interrupt Enable) = 1.
- EXL (Exception Level) = 0.
- ERL (Error Level) = 0.

If these conditions are met, interrupts are recognized according to the setting of the INT and SW mask bits.

Processor Modes – CW4011 processor mode definitions are as follows:

- The processor is in User mode when KSU is equal to 0b10, and EXL and ERL are cleared.
- The processor is in Kernel mode when KSU is equal to 0b00, or either EXL or ERL is set.

Kernel Address Space Access – Access to the Kernel address space is allowed only when the processor is in Kernel mode, that is:

- KSU is equal to 0b00, or either EXL or ERL is set.

User Address Space Access – Access to the User address space is always allowed.

Cold Reset – The contents of the Status register are undefined after a cold reset, except for the following bits:

- ERL and BEV are set.

Warm Reset – The contents of the Status register are unchanged by warm reset, except for the following bits:

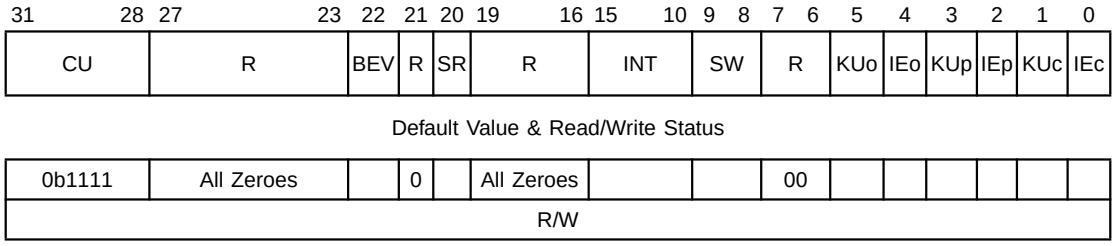
- ERL, BEV, and SR bits are set.

4.6.2.5 Status Register (12) - R3000 Mode

The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. There are two versions of the Status register; one for the R4000 compatibility mode of the CW4011 and one for the R3000 compatibility mode. The compatibility mode is determined by the CMP bit (24) of the CCC register (page 4-74).

The format of the R3000 version of the Status register (CCC[24] = 1) is shown in Figure 4.18.

Figure 4.18 Status Register (R3000 Mode)



CU[3:0] **Coprocessor Enables** **[31:28]**
Each bit controls a coprocessor enable/disable function. For the L64364, all four bits must be set, which enables all four coprocessor inputs.

R **Reserved** **[27:23]**
Not used in L64364. Clear this bit when writing to this register.

BEV **Exception Vector** **22**
Controls the location of general exception vectors. When cleared, the location is normal (0x8000.0000); when set, the location is the bootstrap (0xBFC0.0000). See Section 4.6.3.3, “Exception Vector Locations.”

R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**21**
SR	**Soft Reset** When set, this bit indicates that a soft reset has occurred.	**20**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**[19:16]**
INT[5:0]	**Hardware Interrupt Mask** When set, enables the hardware interrupt mask; when cleared, disables the hardware interrupt mask.	**[15:10]**
SW[1:0]	**Software Interrupt Mask** When set, enables the software interrupt mask. When cleared, disables the software interrupt mask.	**[9:8]**
R	**Reserved** Not used in L64364. Clear this bit when writing to this register.	**[7:6]**
KUo	**Kernel/User Mode (old)** This bit shows the old base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.	**5**
IEo	**Interrupt Enable (old)** When set, interrupts were enabled; when cleared, the interrupts were disabled.	**4**
KUp	**Kernel/User Mode (previous)** This bit shows the previous base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.	**3**
IEp	**Interrupt Enable (previous)** When set, interrupts were enabled; when cleared, interrupts were disabled.	**2**

KUc	**Kernel/User Mode (current)** This bit shows the current base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.	**1**
IEc	**Interrupt Enable (current)** When set, interrupts are enabled; when cleared, interrupts are disabled.	**0**

Interrupt Enable – Interrupts are enabled when IEc is set. Interrupts are recognized according to the state of the INT and SW mask bits. The IEo/IEp/IEc bits form a three-level stack showing the old, previous, and current interrupt enable settings.

Processor Modes – The CW4011 processor is in User mode when KUc is set and in Kernel mode when KUc is cleared. The KUo/KUp/KUc bits form a three-level stack showing the old, previous, and current processor state settings.

Kernel Address Space Access – Access to the Kernel address space is allowed only when the processor is in Kernel mode.

Warm Reset – The contents of the Status register are unchanged by warm reset, except for the following bits:

- BEV and SR bits are set.
- KUo/IEo <- KUp/IEp <- KUc/IEc <- 0/0.

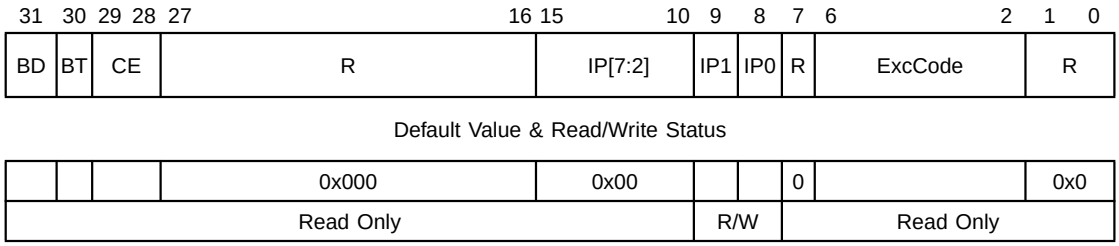
Exception Processing – When an exception is recognized, both the KU and IE bits are “pushed” deeper into the stack with KUc and IEc cleared (i.e., KUo/IEo <- KUp/IEp <- KUc/IEc <- 0/0).

When a Return From Exception (RFE) instruction is executed, the values are “popped” off the stack with KUc/IEc being set to their previous values (i.e., KUc/IEc <- KUp/IEp <- KUo/IE0).

4.6.2.6 Cause Register (13)

The Cause register (described in Figure 4.19 and the paragraphs following it) is a read/write register. It contains information, such as the cause of the most recent exception.

Figure 4.19 Cause Register



BD **Branch Delay** **31**
When set, this bit indicates that the last exception was taken in a branch delay slot.

BT **Branch Taken** **30**
If the Branch Delay bit is set, then the BT bit indicates that a branch was taken when set, and the branch was not taken when cleared.

CE[1:0] **Coprocessor Exception** **[29:28]**
These bits specify the coprocessor unit number that was referenced when a Coprocessor Unusable Exception is taken.

R **Reserved** **[27:16]**
Not used in the L64364. Clear this bit when writing to this register.

IP[7:2] **Interrupt Pending** **[15:8]**
When set, these bits indicate that an External Nonvectored Interrupt is pending, e.g., bit 15 set indicates that IP7 is pending.

IP[1:0] **Software Interrupt Pending** **[9:8]**
When set, these bits indicate that a software Interrupt is pending.

R **Reserved** **7**
Not used in L64364. Clear this bit when writing to this register.

ExcCode[4:0] Exception Code [6:2]

This 5-bit code specifies the cause of the last exception. The exception codes and their descriptions are provided in Table 4.26.

Table 4.26 Exception Codes

Exception Code Value	Mnemonic	Description
0x00	Int	Interrupt
0x01	MOD	TLB Modification exception
0x02	TLBL	TLB exception (load or instruction fetch)
0x03	TLBS	TLB exception (store)
0x04	AdEL	Address error exception (load or instruction fetch)
0x05	AdES	Address error exception (store)
0x06	Bus	Bus error exception
0x07	–	Reserved
0x08	Sys	Syscall exception
0x09	Bp	Breakpoint exception
0x0A	RI	Reserved instruction exception
0x0B	CpU	Coprocessor unusable exception
0x0C	Ov	Arithmetic overflow exception
0x0D	Tr	Trap exception
0x0E	–	Reserved
0x0F	FPE	Floating-point exception
0x10–0x1F	–	Reserved

R Reserved [1:0]

Not used in the L64364. Clear this bit when writing to this register.

All bits in the register, with the exception of the IP[1:0] bits, are read only. The IP[1:0] bits are used for software interrupts.

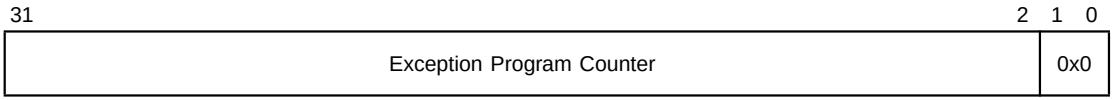
4.6.2.7 Exception Program Counter Register (14)

The Exception Program Counter (EPC) register, shown in Figure 4.20, is a read-write register that contains the address where processing resumes after an exception has been serviced. For synchronous exceptions, the EPC register contains either:

- The virtual address of the instruction that was the direct cause of the exception, or
- The virtual address of the branch or jump instruction (when the instruction is in a branch delay slot, and the Branch Delay (BD) bit in the Cause register is set) immediately preceding the instruction that caused the exception.

Bits 1 and 0 of the register must be cleared.

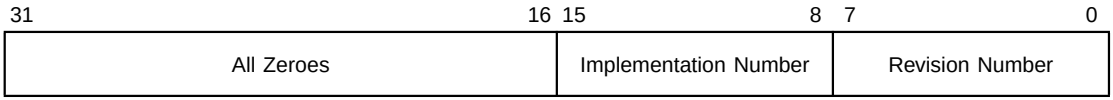
Figure 4.20 EPC Register



4.6.2.8 Processor Revision Identifier Register (15)

The Processor Revision Identifier (PRId) register, shown in Figure 4.21, is a 32-bit, read-only register that contains information identifying the implementation and revision level of the CPU.

Figure 4.21 PRId Register



The low-order byte (bits [7:0]) of the PRId register is interpreted as a CPU unit revision number and the second byte (bits[15:8]) is interpreted as a CPU unit implementation number. The contents of the high-order halfword of the register are reserved.

The revision number is a value of the form y.x, where y is a major revision number in bits [7:4] and x is a minor revision number in bits [3:0].

Note: The revision number can distinguish some chip revisions. However, LSI Logic does not guarantee that changes to this

chip will be reflected in the PRId register, or that changes to the revision number reflect real chip changes. For this reason, these values are not listed and software should not rely on the revision number in the PRId register to characterize the chip.

4.6.2.9 Configuration and Cache Control Register (16)

The Configuration and Cache Control (CCC) register allows software to configure various modules in the CW4011 design (for example, the BIU and Cache Controllers). Figure 4.22 shows the format of the CCC register.

Figure 4.22 CCC Register

31 29	28	27	26	25	24	23	22	21	20	19	18	17	16
R	EWP	R	ISR1	EVI	CMP	IIE	DIE	MUL	MAD	TMR	BGE	IE0	IE1
Default Value & Read/Write Status													
All Zeroes		0	1					1			1		
Read/Write													

15 14	13	12	11 10	9	8 7	6	5	4	3	2	1	0
IS[1:0]	DE0	DE1	DS[1:0]	IPWE	IPWS[1:0]	TE	WB	SR0	SR1	IsC	TAG	INV
Default Value & Read/Write Status												
10			01			0						0
Read/Write												

R **Reserved** **[31:29]**
These bits are not used in the L64364 and must be cleared.

EWP **External Write Priority** **28**
This bit defines SC Bus arbitration priority between data reads and writes in the 4-level write buffer. Clearing EWP gives higher priority to data read requests, if the read address does not match any of the write addresses in the write buffer. Setting EWP gives higher priority to data writes.

R	**Reserved** This bit is not used in the L64364 and must be cleared.	**27**
ISR1	**Instruction RAM Enable** When set with IE1, I-Cache bank 1 operates as Instruction RAM.	**26**
EVI	**External Vectored Interrupt Enable** When set, enables external vectored interrupts; when cleared, disables external vectored interrupts.	**25**
CMP	**R3000 Compatibility Mode** When set, this bit enables the R3000 compatibility mode; when cleared, this bit disables R3000 compatibility mode.	**24**
IIE	**I-Cache Invalidate Request** Not used by the L64364. Can be set or cleared.	**23**
DIE	**D-Cache Invalidate Request** Not used by the L64364. Can be set or cleared.	**22**
MUL	**Floating-Point Multiply Unit Enable** When set, enables the hardware Floating-Point Multiply Unit; when cleared, disables the hardware Floating-Point Multiply Unit. This bit must be set to enable rate calculations in the L64364.	**21**
MAD	**Multiplier Accumulate Extension Enable** When this bit and MUL bit are set, it enables the accumulate extensions; when cleared, disables the accumulate extensions. Not used in the L64364. This bit must be cleared.	**20**
TMR	**Timer Facility Enable** When set, enables the timing facility. When the value of the Count register (page 4-64) reaches the value written in the Compare register (page 4-65), interrupt IP7 (bit 15) in the Cause register (page 4-71) is set causing an interrupt on the next execution cycle if the interrupt is enabled. When TMR is cleared, it disables the timing facility and IP7 is used for external nonvectored interrupt 5 (IntPCIErr).	**19**
BGE	**Bus Grant Enable** Not used by the L64364. Can be set or cleared.	**18**

IE0 **Instruction Cache Set 0 Enable** **17**
When set, enables I-Cache Set 0; when cleared, disables I-Cache Set 0.

IE1 **Instruction Cache Set 1 Enable** **16**
When set, enables I-Cache Set 1; when cleared, disables I-Cache Set 1. See also IR1 bit.

IS[1:0] **Instruction Cache Set Size** **[15:14]**
Sets the size of the I-Cache enabled by bits 17 and 16. In the L64364, these bits must be set to 0b10 to set the I-Cache size to 4 Kbytes.

DE0 **Data Cache Set 0 Enable** **13**
When set, enables D-Cache Set 0; when cleared, disables D-Cache Set 0.

DE1 **Data Cache Set 1 Enable** **12**
When set, enables D-Cache Set 1; when cleared, disables D-Cache Set 1.

DS[1:0] **Data Cache Set Size** **[11:10]**
These bits set the size of the D-Cache enabled by bits 13 and 12. In the L64364, these bits must be set to 0b01 to set the D-Cache size to 2 Kbytes.

IPWE **Internal Page Write Enable** **9**
The L64364 does not make use of the CW4011's page decode logic. The state of this bit is ignored.

IPWS[1:0] **Internal Page Write Size** **[8:7]**
Normally, if the IPWE bit (bit 9) is set, then the IPWS bits set the internal page write size. The size can be set to 1, 2, 4, or 8 Kbytes with the values of 0b00, 0b01, 0b10, or 0b11, respectively. The L64364 does not make use of the CW4011's page decode logic. The settings in this field are ignored.

TE **Translation Buffer Enable** **6**
When set, this bit enables the Translation Buffer; when cleared, it disables the Translation Buffer. Clear this bit when writing to this register.

WB **Write Through/Write Back Cache Select** **5**
For cache addresses not mapped by the Translation Buffer, this bit selects either the Write-Through or

Write-Back cache operation. Setting this bit enables cache writeback; clearing this bit enables cache write through.

SR0 **Scratch-Pad RAM Mode Select** **4**
When set, this bit configures the Set 0 D-Cache as a Scratch-Pad RAM. When cleared, this bit enables the D-Cache mode for Set 0. DC0 must be cleared for Scratch-Pad RAM mode.

SR1 **Scratch-Pad RAM Mode Select** **3**
When set, this bit configures the Set 1 D-Cache as a Scratch-Pad RAM. When cleared, this bit enables the D-Cache mode for Set 1. DC1 must be cleared for Scratch-Pad RAM mode.

IsC **Isolate Cache Mode** **2**
When this bit is set, APU stores go to the cache but do not propagate to external memory.

TAG **Tag Test Mode** **1**
When set, load and store operations access the Tag RAMs and can be used for Tag RAM testing. When cleared, the Tag Test mode is disabled. See page 4-58 for more information.

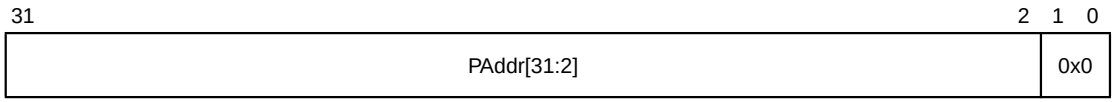
INV **Cache Invalidate Mode** **0**
Set this bit to invalidate the cache contents. Used only for cache diagnostic and debug operations.

4.6.2.10 Load Linked Address Register (17)

The Load Linked Address (LLAdr) register, shown in Figure 4.23, is a read/write register that contains the physical address (PAddr[31:2]) read by the most recent Load Linked instruction. This register is used only for diagnostic purposes and serves no function during normal operation.

The LLAdr register is physically located in the LSU. The CP0 must send read/write signals to the LSU when the value is to be read or written.

Figure 4.23 LLAdr Register

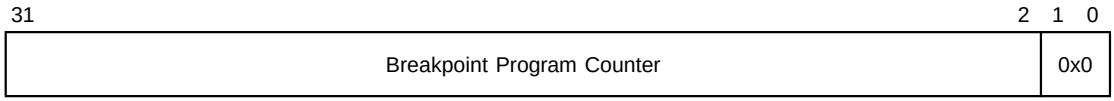


4.6.2.11 Breakpoint Program Counter Register (18)

The Breakpoint Program Counter (BPC) register, shown in Figure 4.24, is a read/write register that software uses to specify a Program Counter breakpoint.

The BPC register is used in conjunction with the Breakpoint PC Mask register, described on page 4-78.

Figure 4.24 BPC Register

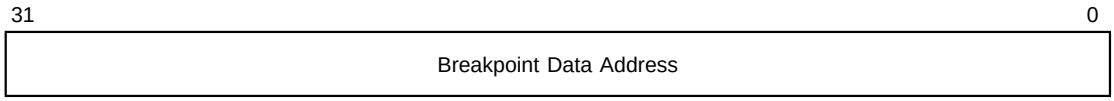


4.6.2.12 Breakpoint Data Address Register (19)

The Breakpoint Data Address (BDA) register (Figure 4.25) is a read/write register that software uses to specify a virtual data address breakpoint.

The BDA register is used in conjunction with the Breakpoint Data Address Mask register, described on page 4-79.

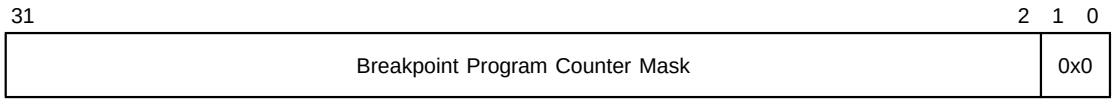
Figure 4.25 BDA Register



4.6.2.13 Breakpoint PC Mask Register (20)

The Breakpoint Program Counter Mask (BPCM) register (see Figure 4.26) is a read/write register that masks bits in the BPC register. A one in any bit in the BPCM register indicates that the CW4011 compares the corresponding bit in the BPC register for Program Counter (debug) exceptions. Values of zero in the mask indicate that the CW4011 does not check the corresponding bits in the BPC register.

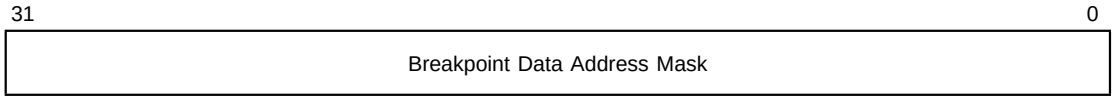
Figure 4.26 BPCM Register



4.6.2.14 Breakpoint Data Address Mask Register (21)

The Breakpoint Data Address Mask (BDAM) register, shown in Figure 4.27, is a read/write register that masks bits in the BDA register. A one in any bit in the BDAM register indicates that the CW4011 compares the corresponding bit in the BDA register for data address (debug) exceptions. Values of zero in the mask indicate that the CW4011 does not check the corresponding bits in the BDA register.

Figure 4.27 BDAM Register



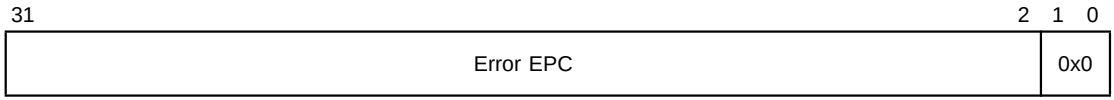
4.6.2.15 Error Exception Program Counter Register (30)

The Error Exception Program Counter (Error EPC) register, shown in Figure 4.28, is similar to the EPC register (page 4-73). It stores the PC on cold cleared, warm cleared, and NMI exceptions. The read/write Error EPC register contains the virtual address at which instruction processing resumes after servicing the interrupt. The address may be either:

- the virtual address of the first instruction terminated by the exception, or
- the virtual address of the branch or jump instruction immediately following the terminated instruction when the terminated instruction is in a branch delay slot.

There is no branch delay slot indication for the Error EPC register.

Figure 4.28 Error EPC Register



4.6.3 CW4011 Exceptions

This section describes each of the CW4011 exceptions, including their cause, handling, and servicing.

4.6.3.1 Exception Handling Overview

To handle an exception, the processor saves the current operating state, enters Kernel mode, disables interrupts, and forces execution of a handler at a fixed address. To resume normal operation, the operating state must be restored and interrupts enabled.

When an exception occurs, the EPC register (page 4-73) is loaded with the restart location at which execution can resume after servicing the exception. The EPC register contains the address of the instruction associated with the exception. Or, if the instruction was executing in a branch delay slot, the EPC register contains the address of the branch instruction that immediately preceded the exception.

R4000 Mode Exception Handling – This is the default mode after a cold reset. The CW4011 processor uses the following mechanisms for saving and restoring the operating mode and interrupt status:

- A single interrupt enable bit (IE), which is located in the Status register (page 4-65).
- An exception level (normal, exception), the EXL bit in the Status register.
- An error level (normal, error), the ERL bit in the Status register.

Interrupts are enabled by setting the IE bit and clearing both the EXL and ERL bits.

R3000 Mode Exception Handling – The R3000 mode of operation is simpler than the R4000 mode. The current processor operating state is always defined by the KUc bit (0 = Kernel, 1 = User). The basic mechanism for saving and restoring the operating state of the processor is the Kernel/User (KU) and Interrupt Enable (IE) stack located in the bottom six bits of the Status register (page 4-68.)

When responding to an exception, the previous mode bits (KUp/IEp) are saved in the old mode bits (KUo/IEo). The current mode bits (KUc/IEc)

are saved in the previous mode bits. Then the current mode bits are both cleared.

After exception processing has completed, the saved state is restored through the use of the RFE instruction. It causes the previous mode bits to be copied back into the current mode bits and the old mode bits to be copied back into the previous mode bits. The old mode bits are left unchanged.

Processing Exceptions – Figure 4.29 through Figure 4.33 depict the basic set of actions taken for each of the major CW4011 exception classes.

Figure 4.29 Cold Reset Exception

```
Random <- TLBENTRIES - 1
Wired <- 0
CCC <- 0^32
DCS <- 0^32
ErrorPC <- PC
SR <- 0^4 || SR27..23 || 1 || 0 || 0 || SR19..3 || 1 || SR1..0
PC <- 0xBFC0 0000
```

Figure 4.30 Warm Reset, NMI Exceptions

```
ErrorPC <- PC
if (CCC24 = 0) then
    SR <- SR31..23 || 1 || 0 || 1 || SR19..3 || 1 || SR1..0
else
    SR <- SR31..23 || 1 || 0 || 1 || SR19..6 || SR3..0 || 0^2
endif
PC <- 0xBFC0 0000
```

Figure 4.31 Common Exceptions

```
Cause <- BD || BT || CE || 0^12 || Cause_15..8 || 0 || ExcCode
|| 0^2
if ((CCC_24 = 1) | (SR_1 = 0)) then
    EPC <- PC
endif
if (CCC_24 = 0) then
    SR <- SR_31..2 || 1 || SR_0
else
    SR <- SR_31..6 || SR_3..0 || 0^2
endif
if (SR_22 = 1) then
    if (CCC_24 = 0) then
         PC <- 0xBFC0 0200 + vector offset
    else
         PC <- 0xBFC0 0100 + vector offset
    endif
else
    PC <- 0x8000 0000 + vector offset
endif
```

Figure 4.32 Debug Exception

```
DCS <- DCS_31..6 || T || W || R || DA || PC || DB
Cause <- BD || BT || Cause_29..0
if ((CCC_24 = 1) | (SR_1 = 0)) then
    EPC <- PC
endif
if (CCC_24 = 0) then
    SR <- SR_31..2 || 1 || SR_0
else
    SR <- SR_31..6 || SR_3..0 || 0^2
endif
if (SR_22 = 1) then
    if (CCC_24 = 0) then
         PC <- 0xBFC0 0200 + vector offset
    else
         PC <- 0xBFC0 0100 + vector offset
    endif
else
    PC <- 0x8000 0000 + vector offset
endif
```

Figure 4.33 External Vectored Interrupt Exception

```
Cause <- BD || BT || Cause31..0
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC <- PC
endif
if (CCC24 = 0) then
    SR <- SR31..2 || 1 || SR0
else
    SR <- SR31..6 || SR3..0 || 02
endif
PC <- EXVAp31..2 || 02
```

4.6.3.2 Precision of Exceptions

Exceptions are logically precise. The instruction that causes an exception and all those that follow it are aborted, generally before committing any state, and can be re-executed after servicing the exception. When the instructions that follow an exception are killed, the exceptions associated with those instructions are also killed. This means that exceptions are not taken in the order detected but in instruction fetch order.

In some cases, the characteristics of the pipeline staging cannot guarantee that all states in the processor and associated system remain completely unchanged because of the possibility of not completely executing instructions that follow the exception. Examples of these state changes are:

- Instructions may be read from memory and loaded into the instruction cache.
- The multiply/divide registers (HI and LO) may have been altered by a MULT/MULTU, DIV/DIVU, or MTHI/MTLO instruction.

Normally, the above effects can be ignored because enough of the machine state is restored to allow execution to successfully resume after servicing the exception.

4.6.3.3 Exception Vector Locations

The Cold Reset and NMI exceptions are always vectored to location 0xBFC0.0000. Addresses for other exceptions are a combination of a vector offset and a base address, which is determined by the BEV bit of

the Status register (page 4-65 and page 4-68). Table 4.27 shows the vector base addresses and Table 4.28 shows the vector offsets.

Table 4.27 Exception Vector Base Addresses

BEV	R4000 Mode (CCC[24] = 0)	R3000 Mode (CCC[24] = 1)
0	0x8000.0000	0x8000.0000
1	0xBFC0.0200	0xBFC0.0100

The virtual memory address from the table above may be further remapped to different physical addresses using the APU_ExcMap field in the APU_AddrMap register (page 4-99).

Exception vectors are stored in Secondary EPROM when the BEV bit is set.

Table 4.28 Exception Vector Offset Addresses

Exception	R4000 Mode (CCC[24] = 0)	R3000 Mode (CCC[24] = 1)
TLB Refill	0x00	0x00
Debug	0x040	0x040
All Others	0x180	0x080

4.6.3.4 Exception Priorities

While more than one exception can occur for a single instruction, only the highest priority exception is reported. Table 4.29 shows the priority order of the exceptions with Cold Reset having the highest priority.

Table 4.29 Exception Priority Order

Exception
Cold Reset
NMI
Address Error - Instruction Fetch
Bus Error
Integer Overflow, Trap, System Call, Breakpoint
Reserved Instruction, Floating-Point Error, Coprocessor Unusable
Address Error - Data Access
External Vectored Interrupt
Debug

4.6.3.5 Cold Reset Exception

The Cold Reset exception occurs when the PCI_RSTn signal is asserted and then deasserted or the XPP_APU_RESET bit in the XPP_Ctrl register (page 9-22) is set and then cleared. This exception is not maskable.

The CPU provides a special interrupt vector (0xBFC0.0000) for the Cold Reset exception. The reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the Translation Lookaside Buffer or the cache to handle the exception. The processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

The contents of all registers in the CPU are undefined when the Cold Reset exception occurs except for the following:

- In the Status register (page 4-65 and page 4-68), the Cu[3:0] and SR bits are cleared and the ERL and BEV bits are set. Other bits are undefined.

- The Wired register is initialized to 0x00.
- The Random register is initialized to the value of its upper bound, 0x1F.

Note: Refer to the LSI Logic *MiniRISC CW4011 Superscalar Microprocessor Core Technical Manual* for a description of the Wired and Random registers.

The Cold Reset exception is serviced by initializing all processor registers, coprocessor registers, caches, and the memory system. Servicing is accomplished by performing diagnostic tests, and by bootstrapping the operating system.

4.6.3.6 Nonmaskable Interrupt Exception

The Nonmaskable Interrupt (NMI) exception occurs in response to the falling edge of the SYS_NMIn signal. As the name implies, the NMI exception is not maskable and occurs regardless of the settings of the EXL, ERL, and IE bits in the Status register (page 4-65).

The reset exception vector (0xBFC0.0000) is also used for this exception. The reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to handle the NMI interrupt. The SR bit of the Status register (page 4-65 and page 4-68) is set to differentiate the NMI exception from a Cold Reset exception.

Because an NMI could occur in the midst of another exception, in general it is not possible to continue program execution after servicing an NMI.

Unlike the Cold Reset exception but like other exceptions, the NMI exception is taken only at instruction boundaries. The state of the caches and memory system are preserved by this exception.

The contents of all registers in the CPU, except for the following, are preserved when the NMI exception occurs.

- The Error EPC register (page 4-79), which contains the restart Program Counter.
- The BEV and SR bits of the Status register (page 4-65 and page 4-68), which are set.

- R4000 mode, in which the ERL bit in the Status register (page 4-65) is set.
- R3000 mode, in which KUo/IEo <- KUp/IEp <- KUc/IEc <- 0/0 in the Status register (page 4-68).

The NMI exception is serviced by saving the current processor state for diagnostic purposes and reinitializing the system in a manner similar to that of the Cold Reset exception.

4.6.3.7 Address Error Exception

The Address Error exception occurs when an attempt is made to:

- Load, fetch, or store a word that is not aligned on a word boundary.
- Load or store a halfword that is not aligned on a halfword boundary.
- Load or store a doubleword that is not aligned on a doubleword boundary.

The Address Error exception is not maskable and it uses the common exception vector.

The ExcCode field in the Cause register (page 4-71) specifies the exception cause. For example, when it contains the code AdEL (0x04), it means a data load or instruction fetch caused the exception. If the ExcCode field contains AdES (0x05), it means a data store operation caused the exception.

When the Address Error exception occurs, the BadVAddr register retains the virtual address that was not properly aligned or that attempted to access protected address space. The contents of the VPN field of the Context and EntryHi registers are undefined, as are the contents of the EntryLo register.

> Note: Refer to the LSI Logic *MiniRISC CW4011 Superscalar Microprocessor Core Technical Manual* for descriptions of the BadVAddr, Context, EntryHi, and EntryLo registers.

The EPC register (page 4-73) points to the instruction that caused the exception unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points to the preceding branch instruction, and the BD bit of the Cause register (page 4-71) is set.

The process executing at the time should be handed a “segmentation violation” signal. This error is usually fatal to the process that was executing when the exception occurred.

4.6.3.8 Bus Error Exception

The Bus Error exception occurs when signaled by the ATMizer II+ chip for events such as bus time-out and bus parity errors. This exception is not maskable.

In the CW4011, bus errors are asynchronous events with respect to CPU instruction processing (much like an NMI). This means that there is no attempt to identify the instruction that caused the error.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Bus (0x06).

The EPC register (page 4-73) points to the first instruction for which processing did not complete unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register (page 4-71) is set.

The physical address where the fault occurred is not available to the exception handler. The process executing at the time of the exception must be handed a “bus error” signal, which is usually fatal.

4.6.3.9 Integer Overflow Exception

The Integer Overflow exception occurs when an ADD, ADDI, SUB, DADD, DADDI, or DSUBI instruction results in a two’s complement overflow. This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to OV (0x0C).

The EPC register (page 4-73) points to the instruction that caused the exception unless the instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register is set.

The process executing at the time of the exception is handed an integer overflow signal. This error is usually fatal to the current process.

4.6.3.10 Trap Exception

The Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, or TNEI instruction results in a TRUE condition. This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Tr (0x0D).

The EPC register (page 4-73) points to the instruction that caused the exception unless the instruction is in a branch delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register is set.

The process executing at the time of the exception is handed a trap signal. This error is usually fatal.

4.6.3.11 System Call Exception

The System Call exception occurs on an attempt to execute the SYSCALL instruction. This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Sys (0x08).

The EPC register (page 4-73) points to the SYSCALL instruction that caused the exception unless this instruction is in a branch delay slot. If in a branch delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register is set.

When this exception occurs, control is transferred to the applicable system routine. To resume execution, the routine must restart instruction execution after the SYSCALL instruction. This restart address can be computed using the EPC register along with the BD and BT bits held in the Cause register.

- If (BD = 0) then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)) then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)) then Restart_PC = Branch Target Address

It is up to the exception handler to obtain the Branch Target Address from the prior branch when the SYSCALL instruction resides in a Branch Delay slot.

4.6.3.12 Breakpoint Exception

The Breakpoint exception occurs during an attempt to execute the BREAK instruction. This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Bp (0x09).

The EPC register (page 4-73) points to the BREAK instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register is set.

When the Breakpoint exception occurs, control is transferred to the applicable system routine. Additional distinctions can be made from the unused bits of the BREAK instruction (bits [25:6]) by loading the contents of the instruction to which the EPC register points (a value of four must be added to the EPC register to locate the instruction if it resides in a Branch Delay slot).

To resume execution, the routine must restart instruction execution after the BREAK instruction. The restart address can be computed using the EPC register along with the BD and BT bits held in the Cause register.

- If (BD = 0), then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)) then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)), then Restart_PC = Branch Target Address

The exception handler must obtain the Branch Target Address from the prior branch when the BREAK instruction resides in a Branch Delay slot.

4.6.3.13 Reserved Instruction Exception

The Reserved Instruction exception occurs when an attempt is made to execute an instruction whose major opcode (bits [31:26]) are undefined, or a SPECIAL instruction whose minor opcode (bits [5:0]) are undefined. This exception also occurs on a REGIMM instruction whose minor opcode (bits [20:16]) are undefined. This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Reserved Instruction (0x0A).

The EPC register (page 4-73) points to the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction, and the BD bit of the Cause register is set.

The Reserved Instruction exception can be used to trap to the emulation routines for instructions not supported in the CW4011 instruction set. Once emulation has been completed, execution can be resumed using the EPC register (page 4-73) along with the BD and BT bits held in the Cause register (page 4-70).

- If (BD = 0), then Restart_PC = EPC + 4
- If ((BD = 1) and (BT = 0)), then Restart_PC = EPC + 8
- If ((BD = 1) and (BT = 1)), then Restart_PC = Branch Target Address

The exception handler must obtain the Branch Target Address from the prior branch when the instruction receiving a Reserved Instruction exception resides in a Branch Delay slot.

If there is no emulation routine, the process executing at the time of the exception should be given an illegal instruction signal. This error is usually fatal.

4.6.3.14 Floating-Point Exception

A floating-point coprocessor is not included in the L64364. Erroneous floating-point accesses will result in this exception. The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to FPE (0x0F).

The contents of the Floating-Point Control Status register (inside CP1) indicate the cause of this exception.

The EPC register (page 4-73) points to the first instruction for which processing did not complete unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction, and the BD bit of the Cause register is set.

This exception is cleared by clearing the appropriate bit in the Floating-Point Control Status register. For an unimplemented instruction exception, the Kernel should emulate the instruction. For other

exceptions, the Kernel should pass the exception to the user process that caused the exception.

4.6.3.15 Coprocessor Unusable Exception

The Coprocessor Unusable exception occurs when an attempt is made to execute a coprocessor instruction for either:

- A corresponding coprocessor unit that has not been marked usable, or
- CP0 instructions when the unit has not been marked usable and the process is executing in User mode.

This exception is not maskable.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to CPU (0x0B). The contents of the CE field in the Cause register specifies which coprocessor was attempting to be referenced.

The EPC register (page 4-73) points to the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction, and the BD bit of the Cause register is set.

Results are one of the following:

- If the process is entitled to access, the coprocessor is marked usable and the corresponding user state is restored.
- If the process is entitled to access the coprocessor, but the coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.
- If the process is not entitled to access the coprocessor, the process executing at the time should be given an Illegal/Privileged Instruction signal. This error is usually fatal.

4.6.3.16 Interrupt Exception

The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts is dependent upon the specific system implementation. Each of the eight interrupts can be masked by clearing the corresponding bit in the INT (Interrupt Mask) field

of the Status register (page 4-65 and page 4-68). All eight interrupts can be masked at once by clearing the IE bit of the Status register.

The common exception vector is used for this exception. The ExcCode field in the Cause register (page 4-71) is set to Int (0x00).

The IP field of the Cause register indicates the current interrupt requests. More than one of the bits can be set simultaneously. If an interrupt is asserted and then deasserted before this register is read, then none of the bits are set.

The EPC register (page 4-73) points at the first instruction for which processing did not complete unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indication.

If the interrupt is caused by one of the two software generated exceptions, the interrupt condition is cleared by clearing the corresponding Cause register bit.

If the interrupt is hardware generated, the interrupt condition is cleared by correcting the condition that caused the interrupt signal to be asserted.

4.6.3.17 External Vectored Interrupt Exception

The CW4011 implements an external vectored interrupt interface, which consists of the following:

- An interrupt input (EXViNTn)
- Interrupt vector virtual address input (EXVAp[31:2])
- Interrupt accepted output (EXVAEn)

The signals must be asserted/deasserted synchronously to the rising edge of the system clock. This interrupt class can be enabled/disabled by setting/clearing the EVI bit in the CCC register (page 4-74).

An External Vectored Interrupt exception occurs when EXViNTn is asserted. The significance of the interrupt is dependent upon the specific system implementation. The interrupt can be disabled by clearing the IE (R3000 = IEc) bit of the Status register (page 4-65 and page 4-68).

The virtual address specified by EXVAp[31:2] is used to specify the target exception handling routine. The EXVAp[31:2] address must be provided by an interrupt controller in the L64364 (see Section 4.8.2, “External Vectored Interrupt Sources,” page 4-103). The EXViNTn and EXVAp[31:2] inputs must be held stable and valid until the exception is accepted, as indicated by the assertion of the EXVAEn output for one cycle.

The EPC register (page 4-73) points to the first instruction for which processing did not complete unless this instruction is in a branch delay slot. If the instruction is in a Branch Delay slot, the EPC register points to the preceding branch instruction and the BD bit of the Cause register (page 4-71) is set as an indication.

The interrupt condition can be cleared in the user-defined interrupt controller by either:

- detecting the assertion of the interrupt accepted output (EXVAEn), or
- by correcting the condition that caused the interrupt signal (EXViNTn) to be asserted.

4.6.3.18 Debug Exception

The Debug exception occurs when a debug condition (read/write access at Breakpoint Data Address, read access at Breakpoint Program Counter, Trace) is detected by the CP0. The Debug Control and Status (DCS) register (page 4-63) specifies which event was detected.

R4000 Mode – Maskable by setting the EXL bit in the Status register (page 4-65). When set, a debug event does not cause an exception trap even if the TE bit in the DCS register (page 4-63) is set. However, the status bits of the DCS register are updated to indicate an event was recognized.

R3000 Mode – Not maskable.

The Debug exception vector is used to handle this exception.

The Debug exception is a debugging aid. Typically, the exception handler transfers control to a debugger, which allows you to examine the situation. To continue, the debug exception condition must be disabled to execute the faulting instruction and then re-enabled.

Programmer’s Notes:

1. The T (Trace Event Detected) bit in the DCS register (page 4-63) is set whenever a branch instruction is encountered regardless of whether the branch is actually taken or not. However, if the debug exception trap is enabled (TR bit in the DCS register set), an exception is recognized only if the branch is actually taken and the target instruction is executed.
2. The PC bit in the DCS register is set whenever the target address of a branch falls within the specified PC address range (BPC, BPCM) regardless of whether the branch is actually taken or not. However, if the Debug exception trap is enabled, an exception is recognized only if the branch actually is taken and the target instruction is executed.

4.7 Memory Map

This section describes the ATMizer II+ chip memory map from the APU perspective. The CW4011 Core uses a 32-bit address and has an address space of 4 Gigabytes.

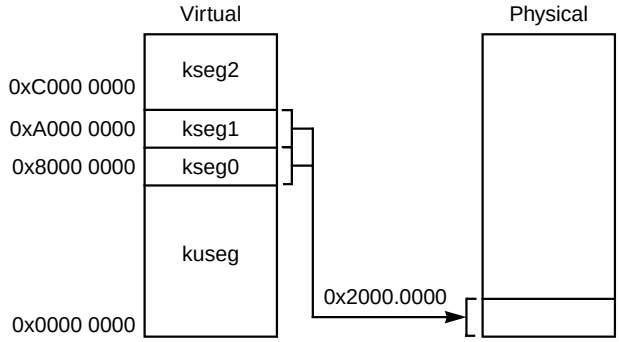
Although the CW4011 Core supports a Memory Management Unit (MMU) with a Translation Lookaside Buffer (TLB) for virtual to physical address translation, the MMU is not implemented in the ATMizer II+ chip.

4.7.1 Operating Modes

The CW4011 Core operates in either a kernel mode or in a user mode, and the address mapping is different in each mode.

In the user mode, the CW4011 has a single uniform address space (kuseg) of 2 Gigabytes available. All memory references in the user mode must have the MSB of the effective address cleared. In the kernel mode, the address space is divided into four regions as shown in Figure 4.34.

Figure 4.34 CW4011 Virtual Memory Map



Without an MMU, the differences between the four memory segments concern the memory mapping and whether the segment is cacheable. Three MSBs of the effective address are used to identify a segment. Note that both kseg0 and kseg1 are mapped to the same physical address. This feature allows access to a memory location in the first 512 Mbytes with cache enabled (kseg0) or not (kseg1).

Table 4.30 shows the properties of the four memory segments. In the following description, the virtual address is the address that is used by the programmer, while the physical address is the address present on the APU Bus.

Table 4.30 Segment Properties

Segment	Size	Virtual Address	Physical Address	Cacheable	Mode
kuseg	2 Gigabytes	0x0000.0000	0x0000.0000	Yes	User/Kernel
kseg0	512 Mbytes	0x8000.0000	0x0000.0000	Yes	Kernel
kseg1	512 Mbytes	0xA000.0000	0x0000.0000	No	Kernel
kseg2	1 Gigabyte	0xC000.0000	0xC000.0000	Yes	Kernel

The ATMizer II+ chip must use the APU in the kernel mode, because all internal and external resources are mapped on kernel segments.

4.7.2 ATMizer II+ Chip Memory Map

All ATMizer II+ chip bus masters (EDMA, APU, and Scheduler) can access internal resources, such as the Cell Buffer Memory or hardware

registers. They can access external resources through the Primary and Secondary Ports. The memory map shown in Table 4.31 is common to all bus masters. The Secondary Bus size can optionally be increased to 64 Mbytes by the APU.

Table 4.31 ATMizer II+ Chip Memory Map

Resource	Size	Virtual Base Address	Physical Base Address	Cache able	Access
Secondary Port	16/64 Mbytes	0x8000.0000	0x0000.0000	Yes	APU
Primary Port	128 Mbytes	0x8800.0000	0x0800.0000	Yes	APU
Secondary Port	16/64 Mbytes	0xA000.0000	0x0000.0000	No	All[1]
Primary Port	128 Mbytes	0xA800.0000	0x0800.0000	No	All[1]
Cell Buffer Memory	16 Kbytes	0xB000.0000	0x1000.0000	No	All[1]
Hardware registers	4 Kbytes	0xB800.0000	0x1800.0000	No	APU
Exception vectors	1 Mbyte	0xBFC0.0000	0xXXX0.0000[2]	No	APU

1. Map for APU, EDMA and Scheduler
2. Depending on APU_Boot and APU_ExcMap bits, see Section 4.6, “Exceptions.”

The APU Data RAM and the APU Instruction RAM are not shown in the memory map; only the APU can access these two memories. The mapping of the APU Data RAM and the APU Instruction RAM is specified at initialization by setting the Tag memories to appropriate values. This is further explained in Section 4.5.3, “D-Cache Scratch-Pad RAM Mode,” and Section 4.5.4, “I-Cache RAM Mode.”

Cycles initiated on an internal bus by either the APU, EDMA, or Scheduler to an undefined memory location terminate with a bus error.

APU exception vectors are located at virtual address 0xBFC0.0000. The 1 Mbyte space starting at that address may be remapped to the Cell Buffer Memory or the Secondary Port so that the ATMizer II+ chip may boot from either location. The Cell Buffer Memory code can be loaded either from the serial interface (i.e., the contents of the serial EPROM) or from a host on the PCI Bus after a reset.

4.7.3 Hardware Registers Map

As shown in Table 4.32, the ATMizer II+ chip hardware register base address is 0x1800.000 (virtual 0xB800.0000). The table provides the base address for registers located in functional modules. The offsets for individual registers are provided in the chapters that describe the respective registers and in Appendix A.

Table 4.32 ATMizer II+ Chip Hardware Register Map

Module	Virtual Base Address	Physical Base Address	Size (Bytes)
EDMA	0xB800.0000	0x1800.0000	256
ACI	0xB800.0100	0x1800.0100	256
Scheduler	0xB800.0200	0x1800.0200	128
Timer	0xB800.0280	0x1800.0280	128
APU	0xB800.0300	0x1800.0300	256
Primary Port	0xB800.0400	0x1800.0400	256
SBC	0xB800.0800	0x1800.0800	1 K
PCI Configuration registers	0xB800.0900	0x1800.0900	256

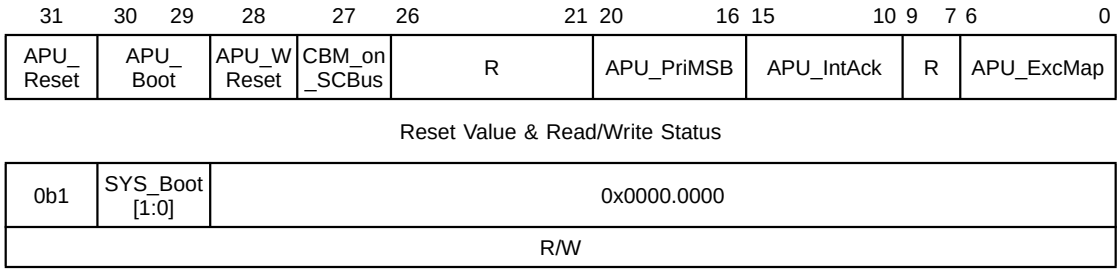
The APU cache memories are not accessible to the external bus masters; the hardware registers are accessible to the external bus masters. The PCI configuration registers are accessible to the APU as well. They are in the address range 0xB800.0900–0xB800.0944. The APU has full read/write access to these registers and accesses them in the big endian format.

4.7.4 ATMizer II+ Chip Primary and Secondary Port Access

The ATMizer II+ chip internal bus masters (APU, EDMA, and Scheduler) directly access external devices through either the PCI or Secondary Bus Controller. The address space on the PCI is 128 Mbytes and, together with Cell Buffer Memory and hardware registers, results in an internal address bus with bits[28:0]. Address bits [28:27] define whether the operation is a Primary or Secondary Port access or an internal access (Cell Buffer Memory or hardware registers).

The APU_AddrMap register, shown in Figure 4.35, holds the most significant bits of an address for an ATMizer II+ chip access through the Primary or Secondary Port. These bits extend the external address buses to 32-bits. The register is located at memory address 0xB800.0300.

Figure 4.35 APU_AddrMap Register



APU_Reset **APU Reset** **31**

The APU_Reset bit is set when the hardware PCI_RSTn signal is asserted. All hardware modules remain in an idle state as long as this bit is set. The APU should initialize all hardware registers and memory resident data structures before clearing the APU_Reset bit. The APU can set the bit during operations, effectively performing a software reset.

The EDMA, ACI, Scheduler, and Timer registers and most APU registers are cleared on the first clock cycle after APU_Reset is set. The registers are then programmed to their default values and held in that state until APU_Reset is cleared. APU_Reset should be cleared before cell commands are issued to the EDMA.

If either the SC Bus or OCA Bus watchdog times out while APU_Reset is set, a boot fault condition is entered (see the XPP_BootFault bit on page 9-23).

APU_Boot **APU Boot Code Location** **[30:29]**

The APU_Boot bits are copied from SYS_BOOT[1:0] signals when PCI_RSTn is deasserted. The bits specify where the exception vector virtual address 0xBFC0.0000 is mapped. This is further described in Section 4.11.1, “Boot Location.” The APU can change these bits during run time, since it can write to the APU_AddrMap register.

APU_WReset

PCI Host Warm Reset **28**

The APU_WReset bit is set to indicate that the last reset was due to an XPP_APU_WRest.

CBM_on_SCBus

CBM Accesses on SC Bus **27**

This bit is set to indicate when the CBM is accessed over the SC Bus and cleared when the CBM is accessed over the OCA Bus.

Note: Signed halfword and byte accesses to the CBM over the OCA Bus are not supported. Unsigned accesses of all sizes are supported.

R **Reserved** **[26:21]**

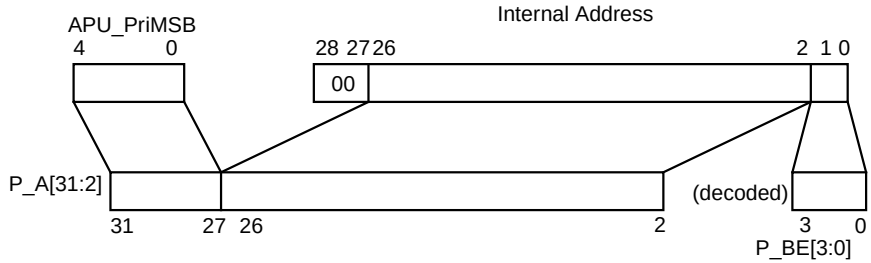
Not used in the L64364.

APU_PriMSB

Primary Port Address MSB **[20:16]**

The 5-bit wide field, APU_PriMSB, is concatenated with the 25-bit wide internal address to form a 32-bit wide external address (bits[1:0] are don't cares) on the Primary Port. The mapping is illustrated in Figure 4.36. Wait until all accesses to the port are completed before updating this field.

Figure 4.36 Primary Port Address Formation



APU_IntAck[5:0]

APU Interrupt Acknowledge **[15:10]**

The 6-bit field, APU_IntAck, resets the corresponding bit of the nonvectored interrupts described in Section 4.8.1, "External Nonvectored Interrupts." Pending interrupts are cleared by setting the corresponding bit in the

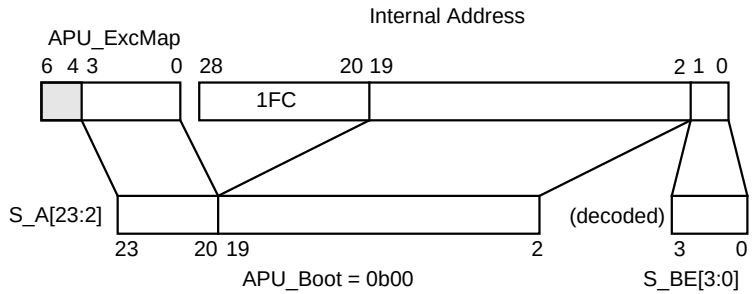
APU_IntAck field. The APU Cause register (page 4-70), contains the status of nonvectored interrupts.

R **Reserved** **[9:7]**
Not used in the L64364.

APU_ExcMap

Exception Vector Address MS Bits **[6:0]**
The exception vector space located at virtual address 0xBFC0.0000 may be further remapped using the APU_ExcMap field as shown in Figure 4.37. This space is mapped to the Secondary Port when the APU_Boot field (refer to page 4-118) is 0b00.

Figure 4.37 Secondary Address Formation for Exception Vectors



4.8 Interrupts

The CW4011 processor used in the ATMizer II+ chip supports three types of interrupt signals:

- Cold/warm resets (CRESETn and WRESETn signals) and nonmaskable interrupts (NMIn signal)
- External interrupts (EXiNTn[5:0])
- External vectored interrupts (EXiNTn)

4.8.1 External Nonvectored Interrupts

The CW4011 Processor has six nonvectored Interrupts. These interrupts are used internally in the L64364 to process catastrophic events.

Table 4.33 lists and defines the nonvectored interrupts. See also Section 4.6, “Exceptions.”

Table 4.33 Nonvectored Interrupt Sources

Name	Number	IP Bit in Cause Reg.[1]	CW4011 Status Reg. Bit[2]	Description
IntPCIErr	5	7	15	PCI abort or parity error
IntSBErr	4	6	14	Secondary Bus error
IntRateExc	3	5	13	Rate calculation exception or OCA Bus time-out
IntRxMbxOvr	2	4	12	Receive Mailbox overflow
IntSCD_BusErr	1	3	11	Scheduler bus error
IntEDMA_BusErr	0	2	10	EDMA bus error

1. See page 4-71.
2. See page 4-65.

Nonvectored interrupts are serviced by three registers. The interrupts are reported using the IP field (bits [15:10]) of the CW4011 Cause register, individually enabled/disabled using the Int field (bits [15:10]) of the CW4011 Status register, and cleared using the APU_IntAck field (bits [15:10]) of the APU_AddrMap register.

Although nonvectored interrupts are cleared by setting the corresponding APU_IntAck bit in the APU_AddrMap register, this only clears the interrupt and not the error condition. Bad data (parity error) or bad pointers (address error) may still exist in the system.

4.8.1.1 IntPCIErr

Sets when a target abort, master abort, or data parity error occurs on the PCI Bus.

4.8.1.2 IntSBErr

Sets on an SB_RDYn time-out or when an access is attempted to:

- a Secondary Memory page that is not enabled.

- an address greater than 16 Mbytes (0x0100.0000) or 64 Mbytes (0x0400.0000). See Section 4.7.2, “ATMizer II+ Chip Memory Map.”

4.8.1.3 IntRateExc

Sets when a rate instruction (`rmul`, `rsub`, or `radd`) result cannot be expressed in the target format or the OCA watchdog timer times out. The cause of this interrupt can be determined by reading the `OCAbus_TimeOut` bit in the APU_Error register, which is set when the OCA watchdog timer times out.

4.8.1.4 IntRxMbxOvr

Sets when a PCI master attempts to write to a full Receive Mailbox FIFO. The write data is dropped and the `IntRxMbxOvr` interrupt is asserted to the APU. The interrupt is cleared by reading the Receive Mailbox.

4.8.1.5 IntSCD_BusErr and IntEDMA_BusErr

Set when a bus error is asserted during a Scheduler or EDMA bus cycle. Since these modules are not capable of handling bus errors, the exception is passed to the APU. To clear the interrupt, set the corresponding `APU_IntAck` bit in the APU_AddrMap register (page 4-99).

Note: When `EDMA_BusErr` is signaled, the EDMA module will no longer process any requests since its control structures may have been corrupted. This condition can only be cleared by asserting `APU_Reset`.

When `SCD_BusErr` is signaled, the Scheduler module will not reliably complete any further scheduler commands. This condition can only be cleared by asserting `APU_Reset`.

4.8.2 External Vectored Interrupt Sources

The ATMizer II+ chip integrates an interrupt controller to efficiently support external vectored interrupts. Possible sources of vectored interrupts are listed in Table 4.34. Interrupt number 15 has the highest

priority and interrupt number 0 has the lowest priority. The interrupts are level sensitive. (See also Section 4.3.7, “Coprocessor Instructions.”)

Table 4.34 Vectored Interrupt Sources

Name	Number[1]	Description
IntEDMA_ComplFull	15	TxCell, RxCell or Buff Completion Queue is full
IntACI_RxFull	14	ACI Receive FIFO full
IntRxMbx	13	Receive Mailbox FIFO nonempty
IntMove_Compl	12	Move complete
IntEDMA_RxCompl	11	RxCell Completion Queue nonempty
IntACI_RxThrld	10	ACI Receive FIFO exceeds threshold
IntEDMA_TxCompl	9	TxCell Completion Queue nonempty
IntEDMA_BuffCompl	8	Buff Completion Queue nonempty
IntACI_Err	7	Tiimeout, parity, or short-cell error
IntACI_TxThrld	6	ACI Transmit FIFO drops below threshold
IntExt[1:0]	5–4	External interrupt inputs (user-defined)
IntTim[3:1]	3–1	Timers 3–1 time-out
IntTim[0]	0	Timer 8 time-out

1. APU Status register interrupt bit numbers and APU_VIntEnable register bit numbers match the interrupt numbers shown here.

4.8.2.1 IntEDMA_ComplFull

This condition occurs when either the EDMA TxCell, RxCell, or Buff Processor Completion Queue is full. The status of which completion queue is full is indicated in the APU_Status register. The EDMA processors are stalled when they attempt to write to a full completion queue. The corresponding interrupt is cleared when the Tx, Rx, or Buff Completion Queue is read.

4.8.2.2 IntACI_RxFull

This condition occurs when the ACI Receiver requests a free cell location from the Cell Buffer Manager and there is none available or the ACI

Receiver FIFO is full. The ACI Receiver FIFO is full when the number of cells in the FIFO exceeds the value programmed in the ACI_RxSize register. The ACI Receiver stops cell reception at that time, resulting in possible cell loss due to the physical device buffer overflow. The interrupt is cleared when the ACI Receiver resumes normal operations, that is, when free cells become available or the FIFO holds less than ACI_RxSize cells. Refer to Section 6.6, “ACI Receiver,” for ACI Receiver operation.

4.8.2.3 IntRxMbx

This interrupt is set when the Mailbox Receive FIFO (Primary Port master to APU) holds one or more messages and is cleared when the FIFO is empty.

4.8.2.4 IntMove_Compl

This interrupt is set whenever the `ComStatus` field of the EDMA_MoveCount register or the `ComStatus2` field of the EDMA_MoveCount2 register has a nonzero value. Note that, to clear a bit in the `ComStatus` field, a ‘1’ must be written to the bit position. Refer to Section 5.3.4, “Move Command,” for more details on this register.

4.8.2.5 IntEDMA_RxCompl, IntEDMA_TxCompl, and IntEDMA_BuffCompl

These interrupts are set when the corresponding EDMA processor completion queue holds one or more completion messages. The APU should drain the completion queues at a sufficiently high rate to avoid EDMA processor stalls. The interrupts are cleared when the corresponding completion queues are emptied.

4.8.2.6 IntACI_RxThrld

The interrupt is set when the number of cells in the ACI Receive FIFO exceeds the value programmed in the ACI_RxLimit register. The interrupt is cleared when the number of cells drops below the programmed threshold.

4.8.2.7 IntACI_TxThrld

The interrupt is set when the number of cells in the ACI Transmit FIFO drops below the value programmed in the ACI_TxLimit register and is cleared when the FIFO size exceeds the programmed threshold.

4.8.2.8 IntACI_Error

The interrupt is set whenever there is a time-out error on the ACI Transmit, or a parity error or short-cell error on the ACI Receive.

4.8.2.9 IntExt[1:0]

These are the inputs to the L64364 for user-defined external interrupts.

4.8.2.10 IntTim[3:0]

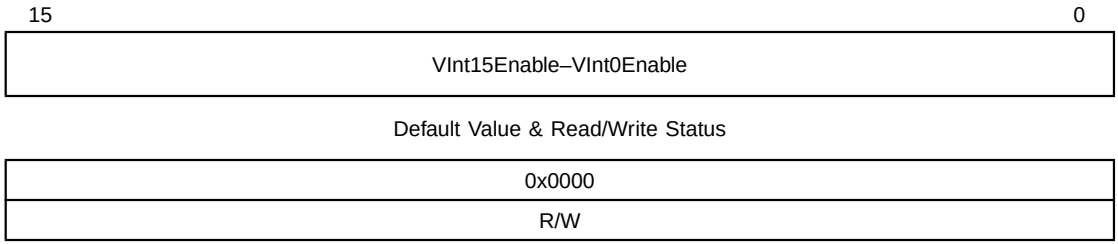
An enabled timer time-out event generates a corresponding `IntTim[3:0]` interrupt (see Section 8.1, “Introduction” and Section 8.3, “Time-Out Events”). The timer interrupts are cleared when the APU writes to the TM_Clear register. `IntTim[3:1]` are set when the general purpose Timers 3 to 1, respectively, reach time-out. `IntTim[0]` is set when Timer 8 reaches time-out.

4.8.3 Enabling Vectored Interrupts

The vectored interrupts must be globally enabled by setting the `EVI` bit in the APU CCC register (page 4-74).

Each interrupt listed in Table 4.34 may also be individually enabled using the APU_VIntEnable register located at memory address 0xB800.030E. The register bits are assigned to interrupts in the same order or the interrupt priority in Table 4.34. Setting a bit in the register enables the corresponding interrupt and clearing a bit disables (masks) the interrupt. Figure 4.38 shows the format of the register.

Figure 4.38 APU_VIntEnable Register



The APU_VIntEnable register is cleared at system reset.

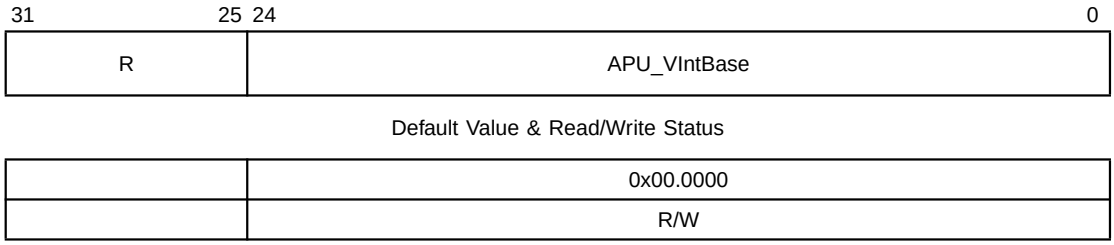
To generate interrupts 0–3, time-out events must be enabled by setting the appropriate bits of the TM_Enable register (see Table 8.1).

4.8.4 Vectored Interrupt Processing

When an enabled vectored interrupt is asserted, the external vectored interrupt exception is generated and the processor starts executing the EVI handler routine. The routine address is a combination of the APU_VIntBase register (shown in Figure 4.39 and located at address 0xB800.0310) and the interrupt number as follows:

```
TargetAddr[31:7] = APU_VIntBase[24:0]
TargetAddr[6:3]  = IntNum
TargetAddr[2:0]  = 0
```

Figure 4.39 APU_VIntBase Register Format



R **Reserved** **[31:25]**
Not used in the L64364.

APU_VIntBase **[24:0]**
`APU_VIntBase[24:0] = Base_Address[31:0] >> 7` and Base_Address is on a 64 byte boundary, that is, bits [6:0] = 0.

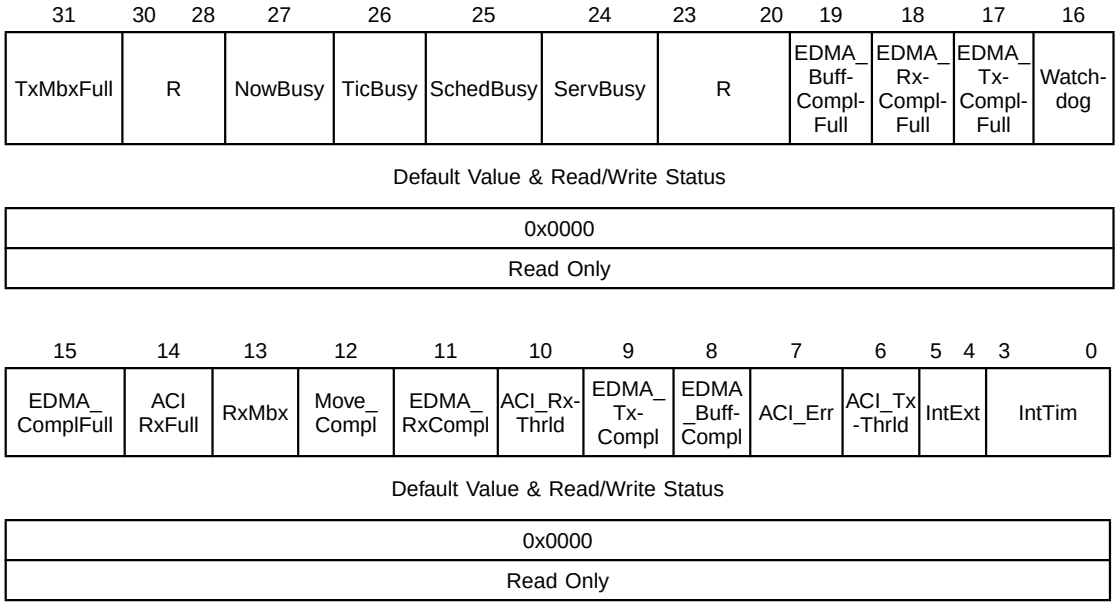
In typical applications, the target address of the external vectored interrupt exception is located in a high-speed memory, such as the APU Scratch-Pad Data RAM or Cell Buffer Memory and contains an absolute jump to the desired handler routine as shown in the example below:

```
j       ComplFullHandler
nop
j       RxFifoFullHandler
nop
j       MboxFullHandler
etc....
```

4.8.5 Status Checking

The APU Status register, shown in Figure 4.40, can be used to check the status of real-time events. Bits [15:0] of the register store the status of the external vectored Interrupts and bits [31:16] store additional events. It is a read-only register and is located at address 0xB800.0314. All bits are cleared at `APU_Reset`.

Figure 4.40 APU Status Register Format



TxMbxFull **Tx Mailbox FIFO Full** **31**
The `TxMbxFull` bit is set when the Mailbox Transmit FIFO (APU to Primary Port) is full.

R **Reserved** **[30:28]**
Not used in the L64364.

NowBusy **Now Command in Process** **27**
The `NowBusy` bit is set when the Scheduler is busy executing a `now` command.

TicBusy **Tic Command in Process** **26**
The `TicBusy` bit is set when the Scheduler is busy executing a `tic` command.

SchedBusy **Sched Command in Process** **25**
The SchedBusy bit is set when the Scheduler is busy executing a sched command.

ServBusy **Serv Command in Process** **24**
The ServBusy bit is set when the Scheduler is busy executing a serv command.

R **Reserved** **[23:20]**
Not used in the L64364.

EDMA_BuffComplFull
Buff Completion Queue Full **19**
The EDMA_BuffComplFull bit is set when the Buff Completion queue is full. This bit is cleared when the completion queue is not full.

EDMA_RxComplFull
Rx Completion Queue Full **18**
The EDMA_RxComplFull bit is set when the Rx Completion queue is full. This bit is cleared when the completion queue is not full.

EDMA_TxComplFull
Tx Completion Queue Full **17**
The EDMA_TxComplFull bit is set when the Tx Completion queue is full. This bit is cleared when the completion queue is not full.

Watchdog **SC Bus Watchdog Timer Time-Out** **16**
The Watchdog bit is set when the SC Bus watchdog timer expires and the current APU Bus transaction is terminated by the bus error. The bit can only be cleared by toggling the APU_Reset bit in the APU_AddrMap register (page 4-99).

EDMA_ComplFull
Tx, Rx, or Buff Completion Queue Full **15**
The EDMA_ComplFull bit is set when either the Tx, Rx, or Buff Completion Queue Full interrupt occurs. It is cleared when the interrupt clears.

ACI_RxFull **No Free Cells Available** **14**
The ACI_RxFull bit is set when an IntACI_RxFull interrupt occurs indicating that the ACI Receiver requested a free cell location from the Cell Buffer

Manager and there is none available, or the number of cells in the ACI_Receive FIFO is above the value in the ACI_RxSize register. It is cleared when the interrupt clears. This bit is also set when the ACI_RxSize register (page 6-17) and ACI_RxCells register (page 6-16) are both zero.

RxMbx **Rx FIFO Not Empty** **13**
The RxMbx bit is set when an IntRxMbx interrupt occurs indicating that the Mailbox Receive FIFO holds one or more messages (Primary Port master to APU). It is cleared when the interrupt clears.

Move_Compl **Move Complete** **12**
The Move_Compl bit is set when a move completion occurs. It is cleared when the interrupt clears.

EDMA_RxCompl
RxCell Completion Queue Not Empty **11**
The EDMA_RxCompl bit is set when an IntEDMA_RxCell interrupt occurs indicating that the RxCell completion queue holds one or more completion messages. It is cleared when the interrupt clears.

ACI_RxThrld **Rx FIFO over Threshold** **10**
The ACI_RxThrld bit is set when an IntACI_Rx interrupt occurs indicating that the number of cells in the ACI Receive FIFO exceeds the value programmed in the ACI_RxLimit register (page 6-9). It is cleared when the interrupt clears.

EDMA_TxCompl
TxCell Completion Queue Not Empty **9**
The EDMA_TxCompl bit is set when an IntEDMA_RxCell interrupt occurs indicating that the TxCell completion queue holds one or more completion messages. It is cleared when the interrupt clears.

EDMA_BuffCompl
Buffer Completion Queue Not Empty **8**
The EDMA_BuffCompl bit is set when an IntEDMA_Buff interrupt occurs indicating that the EDMA buffer completion queue holds one or more completion messages. It is cleared when the interrupt clears.

ACI_Err **Error FIFO Interrupt** **7**

The ACI_Err bit is set when an IntACI_Error interrupt occurs indicating that the ACI Error FIFO holds one or more cells. It is cleared when the interrupt clears.

ACI_TxThrld **Tx FIFO Low Interrupt** **6**

The ACI_TxThrld bit is set when an IntACI_Tx interrupt occurs indicating that the number of cells in the ACI Transmit FIFO dropped below the value programmed in the ACI_TxLimit register (page 6-9). It is cleared when the interrupt clears.

IntExt **External Interrupt** **[5:4]**

The IntExt bits are set to indicate that an external, user-defined interrupt is input to the L64364 on its SYS_INT[1:0] interrupt lines; bit 5 for line 1 and bit 4 for line 0. The bits are cleared when the interrupts clear.

IntTim **Internal Timer Interrupt** **[3:0]**

The IntTim bits are set when an IntTime[3:0] interrupt occurs indicating that general-purpose Timer 3, 2, 1, or 8, respectively, timed out. The bits are cleared by writing a logic 1 to the corresponding bits in the TM_Clear register (page 8-2).

To assure that the APU_IntTim[3:0] bits set when a time-out occurs, time-out events must be enabled by setting the appropriate bits in the TM_Enable register.

Bits [15:0] shows the status of the individual vectored interrupt conditions independent of the setting of the APU_VIntEnable register. The APU_VIntEnable register controls interrupt delivery to the APU but not interrupt reporting in the APU Status register (page 4-108). Time-out registering must be enabled by setting the appropriate bits of the TM_Enable register (page 8-2) for the bits APU_Status[3:0] to be set in case of a time-out event.

There is no way to directly clear the individual bits of the APU Status register (page 4-108) other than asserting APU_Reset. The interrupt conditions that set the bits must be cleared. For instance to clear the APU_ACI_RxFull status bit, the condition that caused the IntACI_RxFull interrupt must be cleared. A cell must be processed and returned to the free cell list. If the interrupt is enabled, its interrupt handling routine forces the cleared condition. Even though an interrupt is masked, its status bit is set when the interrupt condition occurs, and does not clear

until the interrupt condition is normally cleared (without the intervention of the interrupt handler).

4.8.6 Coprocessor Condition Signals

The EDMA Request Queue status bits of the APU Status register (page 4-108) are also available as Coprocessor Condition signals. Table 4.35 lists and defines these signals. CpCond0 is available as input pin SYS_CPCOND. (See also Section 4.3.7, "Coprocessor Instructions.")

Table 4.35 Coprocessor Condition Signals

Signal	Status Bit	Description
CpCond3	EDMA_RxCellFull	RxCell command queue is full
CpCond2	EDMA_TxCellFull	TxCell command queue is full
CpCond1	EDMA_BuffFull	Buffer command queue is full
CpCond0	APU_ExtStat	SYS_CPCOND input pin

4.9 CW4011 OCA Bus Accesses

The OCA Bus of the CW4011 core, instead of the SC Bus, is used to access the on-chip register space and the Cell Buffer memory. This interface allows on-chip modules to be accessed at the Cache Read (CR) stage of the pipeline without going through an SC Bus transaction. Note that a read access on the CW4011 SC Bus has at least a three-clock penalty due to the Bus Interface Unit, and a write access is done through a four-deep write buffer. Thus, if on-chip modules can respond in one cycle, there is no penalty for a read or write cycle.

Note: The CW4011 is the only bus master for the OCA Bus. Instructions cannot be fetched through the OCA Bus.

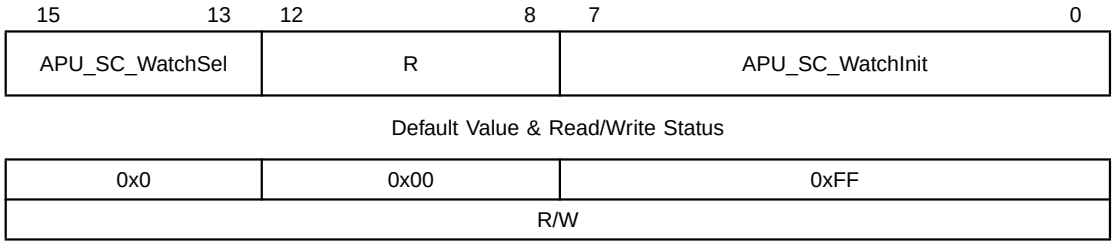
4.10 Bus Watchdog Timers

Since the APU can gain access to internal modules over the SC Bus and the OCA Bus, watchdog timers are implemented for transactions on both buses.

4.10.1 SC Bus Watchdog Timer

The ATMizer II+ chip has an 8-bit watchdog timer for APU transactions. A programmable initial count value for the timer is loaded into the APU_SCbus_Watchdog register (Figure 4.41) when the APU starts a bus transaction. The timer decrements on each system clock pulse or when a general-purpose timer time-out occurs. When the watchdog timer reaches 0, the current APU transaction is terminated by a bus error. This register is located at memory address 0xB800.0306.

Figure 4.41 APU_SCbus_Watchdog Register



APU_SC_WatchSel[2:0]

SC Bus Watchdog Clock Select **[15:13]**

The APU_SC_WatchSel field selects the input clock. Value 0b000 selects the system clock, while values 0b001 through 0b111 select the time-out event of the corresponding general-purpose timer.

R **Reserved** **[12:8]**

Not used in the L64364.

APU_SC_WatchInit[7:0]

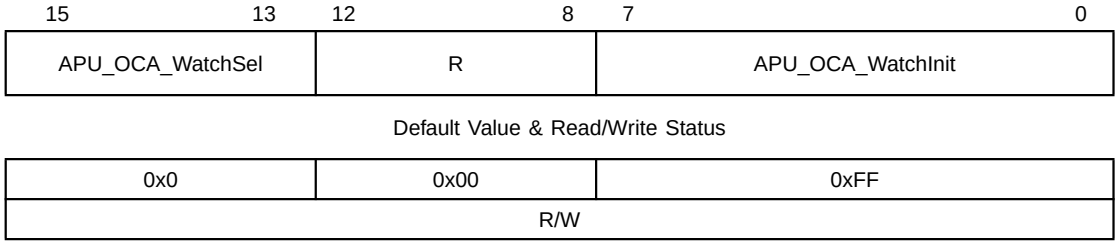
SC Bus Watchdog Initialization **[7:0]**

The APU_SC_WatchInit field is used to specify the watchdog timer initialization value. An initialization value of 0x00 disables the watchdog timer.

4.10.2 OCA Bus Watchdog Timer

This timer takes care of the time-out events on the OCA Bus. The operation is similar to the APU_SCbus_Watchdog register and is shown in Figure 4.42. This register is located at memory address 0xB800.031A.

Figure 4.42 APU_OCAbus_Watchdog Register



APU_OCA_WatchSel[2:0]

OCA Bus Watchdog Clock Select [15:13]

The APU_OCA_WatchSel field selects the input clock. Value 0b000 selects the system clock, while values 0b001 through 0b111 select the time-out event of the corresponding general-purpose timer.

R **Reserved [12:8]**

Not used in the L64364.

APU_OCA_WatchInit[7:0]

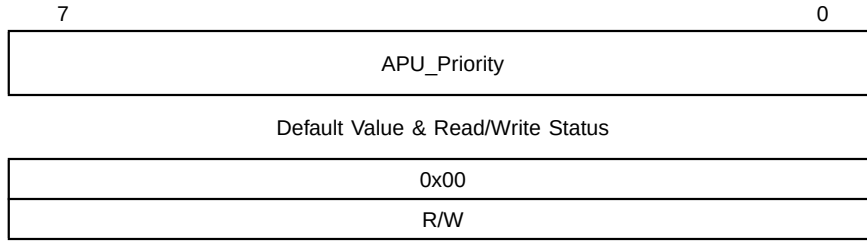
OCA Bus Watchdog Initialization [7:0]

The APU_OCA_WatchInit field is used to specify the watchdog timer initialization value. An initialization value of 0x00 disables the watchdog timer.

4.10.3 APU Priority Register

This 8-bit register implements a timer service. It maintains a stable value and does not change. The value in the APU_SC_WatchInit field of the APU_SCbus_Watchdog register is always compared with the value in this register. Once the value in the APU_SC_WatchInit field equals the APU_Priority register value, APU priority is raised and the APU is granted the next access to the requested bus. Note that this comparison is not enabled until a nonzero value is written to the APU_Priority register. The initialization value of this register is 0x00 and it is located at memory address 0xB800.031F.

Figure 4.43 APU_Priority Register



4.10.4 APU_Error Register

The APU_Error register is used to determine the cause of a bus error due to an APU access and to determine the status of the nonvectored interrupts. The register is located at address 0xB800.0320. The format of the register is shown in Figure 4.44.

Figure 4.44 APU_Error Register

31 16	15	14	13	12	11	10	9 4	3	2	1	0
R	Int PCI Err	Int SB Err	Int Rate-ExcErr	IntRx Mbx-Ovr	IntSCD_ BusErr	IntEDMA_ BusErr	R	Boot_ Fault	Addr_ Error	OCAbus_ TimeOut	SCbus_ TimeOut
Default Value & Read/Write Status											
0x00							0x00		0x0	0x0	
Read Only							Read Only		R/W		

R **Reserved** **[31:16]**
Not used in the L64364.

IntPCIErr **PCI Error Interrupt** **15**
Sets when a target abort, master abort, or data parity error occurs on the PCI Bus.

IntSBErr **SB Error Interrupt** **14**

- Sets when a Secondary Memory page is not enabled.
- Sets when an address greater than 16 Mbytes (0x0100.0000) or 64 Mbytes (0x0400.0000). See Section 4.7.2, “ATMizer II+ Chip Memory Map.”
- SB_RDYn times out on an enabled memory page.

IntRateExcErr

Rate Exception Error Interrupt **13**

Sets when a rate instruction (`rmul`, `rsub`, or `radd`) result cannot be expressed in the target format or the OCA watchdog timer times out. The cause of this interrupt can be determined by reading the `OCAbus_TimeOut` bit in the APU_Error register, which is set when the OCA watchdog timer times out.

IntRxMbxOvr **Rx Mailbox Overflow Interrupt** **12**

Sets when a PCI master attempts to write to a full Receive Mailbox FIFO. The write data is dropped and the `IntRxMbxOvr` interrupt is asserted to the APU. The interrupt is cleared by reading the Receive Mailbox.

IntSCD_BusErr

SCD Bus Error Interrupt **11**

This bit sets when a bus error is asserted during a Scheduler bus cycle. Since the Scheduler is not capable of handling bus errors, the exception is passed to the APU. To clear the interrupt, a ‘1’ must be written to the corresponding `APU_IntAck` bit position in the APU_AddrMap register (page 4-99).

IntEDMA_BusErr

EDMA Bus Error Interrupt **10**

This bit is set when a bus error is asserted during an EDMA bus cycle. Since the EDMA is not capable of handling bus errors, the exception is passed to the APU. To clear the interrupt, a ‘1’ must be written to the corresponding `APU_IntAck` bit position in the APU_AddrMap register (page 4-99).

R **Reserved** **[9:4]**

Not used in the L64364.

Boot_Fault **Boot Fault** **3**

This bit follows the `XPP_BootFault` bit in the XPP_Ctrl register (see page 9-23).

Addr_Error **Address Error** **2**

This bit is set when the APU attempts an access to an undefined address. It is cleared by writing a 1 to the bit position.

OCAbus_TimeOut

OCA Bus Watchdog Timer Time-Out **1**

Sets when the OCA Bus watchdog timer times out. Cleared by writing a 1 to the bit position.

SCbus_TimeOut

SC Bus Watchdog Timer Time-Out **0**

Sets when the SC Bus watchdog timer times out. Cleared by writing a 1 to the bit position.

4.10.5 OCA Error Register

The OCA_Err register is a 32-bit register used to indicate the cause of the OCA watchdog time-out. The register is located at memory address 0xB800.0380. Figure 4.45 shows the format of the register.

Figure 4.45 OCA_Err Register

31	30 29	28	27	26	25 12	11 0
VLD	R	MSTR	R/W	CBM	R	ErrAddr

Default Values & Read/Write Status

0x0		0x0		0x000
R/W	Read Only			

VLD **Valid Data in Register** **31**

This bit is set when the OCA_Err register captures data pertaining to an OCA Bus time-out. Once this bit is set, it must be cleared by writing a ‘1’ to it for the register to capture new time-out data.

R **Reserved** **[30:29]**

The L64364 does not use these bits.

MSTR **OCA Bus Time-Out Master** **28**

When this bit is set, an external PCI host was the OCA Bus master that caused OCA Bus time-out. When this bit is cleared, the APU was the master that caused the OCA Bus time-out.

R/W **Read/Write Access Indicator** **27**

When this bit is set, the OCA Bus time-out occurred on a read access. When cleared, the time-out occurred on a write access.

CBM **CBM Access Indicator** **26**
When this bit is set, the OCA Bus time-out occurred on a Cell Buffer Memory access. When cleared, the time-out occurred on a hardware register access.

R **Reserved** **[25:12]**
The L64364 does not use these bits.

ErrAddr[11:0] **Error Address** **[11:0]**
This field contains the CBM/hardware register address that was accessed when the OCA Bus timed out.

4.11 Boot Procedures

The ATMizer II+ chip APU may boot from Secondary Memory or Cell Buffer Memory (CBM). Two modes for booting from CBM are provided. The first mode permits a PCI master to download the boot code into CBM. In the second mode, the L64364 boot logic copies the boot code from an external device (e.g., a serial EPROM) using the serial interface to CBM.

4.11.1 Boot Location

The APU_Boot bits in the APU_AddrMap register (page 4-99) are copied from the SYS_BOOT[1:0] pins when the PCI_RSTn signal is deasserted. They determine where the 1 Mbyte exception vector space, starting at virtual address 0xBFC0.0000, is remapped as shown in Table 4.36.

Table 4.36 Boot Sequence

SYS_BOOT[1:0] APU_Boot[1:0]	Boot Location	Physical Boot Address
0b00	Secondary Port	0x0000.0000
0b01	Not used	–
0b10	Cell Buffer Memory	0x1000.0000
0b11	Cell Buffer Memory/Serial Interface[1]	0x1000.0000

1. Boot code read from Serial Interface and copied to CBM

4.11.2 Serial Interface Boot Sequence

If SYS_BOOT[1:0] is 0x0B11 when the PCI_RSTn signal is deasserted, the serial interface is used to copy the first 256 bytes from a serial device to the CBM. When the copy is completed, the APU begins the execution of the cold reset exception handler from the CBM. At this point, the CBM contains a maximum of 64 valid instructions. The exception handler could choose to jump to a different boot location or continue downloading boot code from the serial device using the APU_SRL register.

An APU read of the APU_SRL register retrieves the next 32-bit word from the serial device. Certain board- or program-specific information may be stored in these locations and retrieved during normal bootup. The APU_SRL register is located at memory address 0xB800.0308.

4.11.3 Cell Buffer Memory Boot Sequence

If SYS_BOOT[1:0] is 0b10 when the PCI_RSTn signal is deasserted, the boot exception address 0xBFC0.0000 is mapped to CBM address 0. The APU begins the execution of the Cold Reset exception handler from CBM when the XPP_APU_Reset bit in the XPP_Ctrl register (page 9-22) is cleared. The external PCI bus master must download the boot code into CBM before clearing the XPP_APU_Reset bit.

4.11.4 Secondary EPROM Boot Sequence

If SYS_BOOT[1:0] is 0b00 when the PCI_RSTn signal is deasserted, the boot exception address 0xBFC0.0000 is mapped to physical address 0x0000.0000 which is the byte-wide EPROM page of secondary memory. Secondary Bus Control register (page 10-11) initializes to support EPROM access times of 250 ns.

4.11.5 APU Access to Serial EPROM

This feature enables the APU to access serial EPROM, regardless of the Boot mode selected for the APU. Thus, the APU_SRL register can always be used to access a 32-bit word from serial EPROM, irrespective of the SYS_BOOT[1:0] value.

Chapter 5
Enhanced DMA

This chapter explains the operation of the EDMA. It includes the following sections:

- Section 5.1, “Overview,” page 5-1
- Section 5.2, “Data Structures,” page 5-5
- Section 5.3, “EDMA Commands,” page 5-21
- Section 5.4, “Data Structure Locations,” page 5-43
- Section 5.5, “Register Descriptions,” page 5-49
- Section 5.6, “AAL5 Mode Operation,” page 5-57
- Section 5.7, “AAL0 Mode Operation,” page 5-62

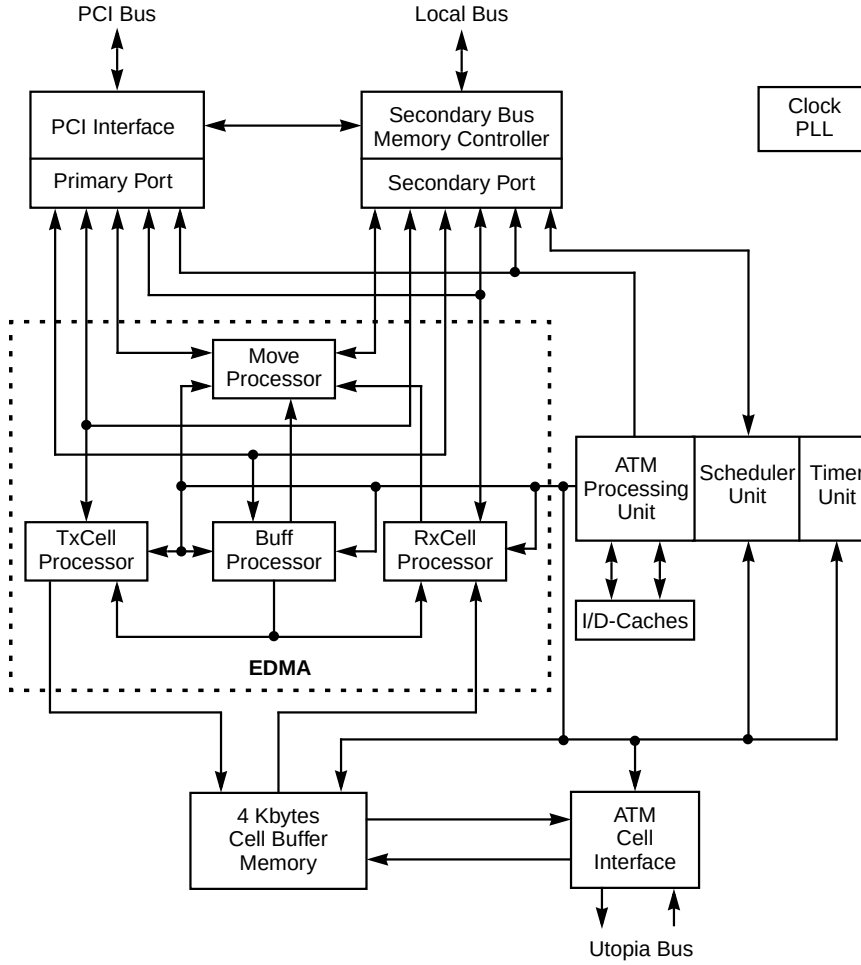
Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” should not be modified.

5.1 Overview

The Enhanced DMA can perform autonomously all SAR related functions on AAL5 CS-PDUs, including creating and extracting the AAL5 CS-PDU trailer. The EDMA also provides significant hardware support for other AALs.

The EDMA is partitioned into four independent processors that can operate in parallel. The four processors, illustrated in Figure 5.1, are the TxCell, RxCell, Buff, and Move Processors.

Figure 5.1 EDMA Processors



The TxCell and RxCell Processors execute transmit and receive `cell` commands. The Buff Processor handles buffer management and executes `buff` commands. The Move Processor is a DMA engine able to transfer data between the Primary and the Secondary Ports.

All processors receive commands from the APU. The Move Processor can also receive commands from the Buff Processor or from the RxCell Processor.

Table 5.1 lists the EDMA processor commands.

Table 5.1 EDMA Commands

Processor	Command	Description
TxCell	`TxCell ConNum, aCell`	Transmit a cell from Connection ConNum using Cell Buffer Memory location aCell
RxCell	`RxCell ConNum, aCell`	Receive a cell from Connection ConNum using Cell Buffer Memory location aCell
TxConClose	`TxConClose ConNum`	Clear VCD_ConOpen for Tx
RxConClose	`RxConClose ConNum`	Clear VCD_ConOpen for Rx
Buff	`Buff BuffNum`	Attach a buffer BuffNum to a Connection Descriptor or a free list
Move	`Move aSrc, aDst, Nbytes`	Move Nbytes of data from source address aSrc to Destination address aDst

The APU controls the operation of the EDMA using hardware registers and memory resident data structures. A write operation to an EDMA hardware register results from a command placed in one of the four EDMA request queues. The TxCell, RxCell, and Buff Processor Request Queues are each four entries deep, while the Move Processor Request Queue is two entries deep. The EDMA processors retrieve commands from their request queues and execute them in order. When a specific action is required from the APU following the completion of a command, the EDMA places a message in one of the three EDMA completion queues. The request and completion queues allow the APU and the EDMA to operate in parallel and to do so without either one stalling while waiting for the other to complete an operation.

A `TxCell` or `RxCell` command contains a Virtual Connection (VC) Descriptor Number of a virtual connection that needs to be serviced and an address of a cell in Cell Buffer Memory. For the receive direction, this is the address of a received cell; for the transmit direction, it is an address of a free location where a cell can be built.

The EDMA computes the address of the VC Descriptor (VCD), reads it to retrieve the necessary information, transfers the data between the source and destination addresses, and updates the VCD. For cells in the transmit direction, the EDMA TxCell Processor optionally can place the

cell in the Transmit FIFO. In that case, for the AAL5 CS-PDU continuation cells, the EDMA performs all the necessary functions and the APU does not need to perform any actions following the transfer completion.

For the AAL5 End-Of-Message (EOM) cells, the APU typically needs to perform housekeeping functions, such as informing the host about the CS-PDU transfer completion. To enable that task, the EDMA TxCell and RxCell Processors store the Buffer Number of the just completed buffer data in the EDMA Transmit or Receive Completion Queue.

The EDMA supports CS-PDUs located in noncontiguous areas of memory (scatter-gather operation). Payload byte alignment is not required for correct operation since the EDMA aligns the bytes during data transfer.

To speed-up buffer management, the EDMA autonomously walks through a linked list of Buffer Descriptors (BFDs). Buffer Numbers of completely processed buffers are returned to the EDMA Transmit or Receive Completion Queues, while the Buffer Numbers of buffers to be processed are retrieved from the EDMA Buffer Request Queue and attached to the end of a linked list of buffers.

For the receive direction, the EDMA maintains twelve lists of free buffers, six pairs of Small and Large Free Buffer Lists. Buffer Numbers are taken from a Free Buffer List as required. However, more sophisticated buffer memory management schemes can be implemented if the APU provides free buffers attached to a VC Descriptor in advance. The EDMA always attempts to use existing buffers before taking one from a Free Buffer List.

VC Descriptors are identified using Connection Numbers. To form the descriptor address, the EDMA adds the Connection Table Base Address (register programmable) to the Connection Number multiplied by the descriptor size. The EDMA supports up to 64 K VC Descriptors. Similarly, to compute Buffer Descriptor addresses, the EDMA adds the Buffer Table Base Address to the Buffer Number multiplied by the Buffer Descriptor size. The EDMA supports up to 64K Buffer Descriptors.

For both VC and Buffer Descriptors, descriptor number 0 is reserved for EDMA internal usage and must not be used.

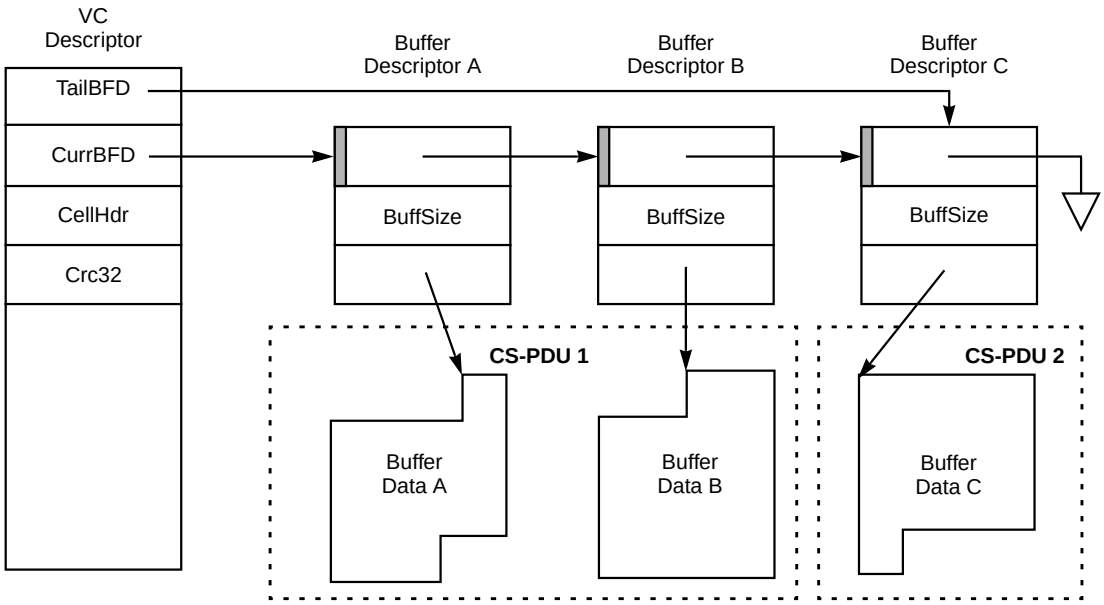
5.2 Data Structures

The EDMA uses two data structures to process receive and transmit cells. The VC Descriptor stores control information about a virtual connection and is typically created when a connection is established. A Buffer Descriptor stores control information about buffer data and is attached to the VC Descriptor when buffer data is segmented or reassembled.

A CS-PDU payload can be placed in one or more buffers. The `BFD_BuffCont` bit located in the control field of a Buffer Descriptor indicates that the CS-PDU payload is continued in a following buffer, pointed to by the `NextBFD` field in the Buffer Descriptor. The `NextBFD` field contains the Buffer Number of the following buffer. This field does not need to be initialized. The EDMA builds the Buffer Descriptor chain in the order of the processed `buff` commands.

Figure 5.2 provides an example of the EDMA data structures with two CS-PDUs. One (called CS-PDU 1) is split between buffers A and B, while the other (called CS-PDU 2) is stored completely in buffer C.

Figure 5.2 Virtual Connection and Buffer Descriptors



CS-PDU payloads are queued for segmentation on a given virtual connection by attaching the corresponding Buffer Descriptors to the tail of the buffer list. The EDMA's Buff Processor performs this operation when it receives a `buff` command from the APU. When buffer data segmentation is completed, the TxCell Processor places the Buffer Number in the EDMA Transmit Completion Queue and removes it from the buffer list by advancing the list head pointer.

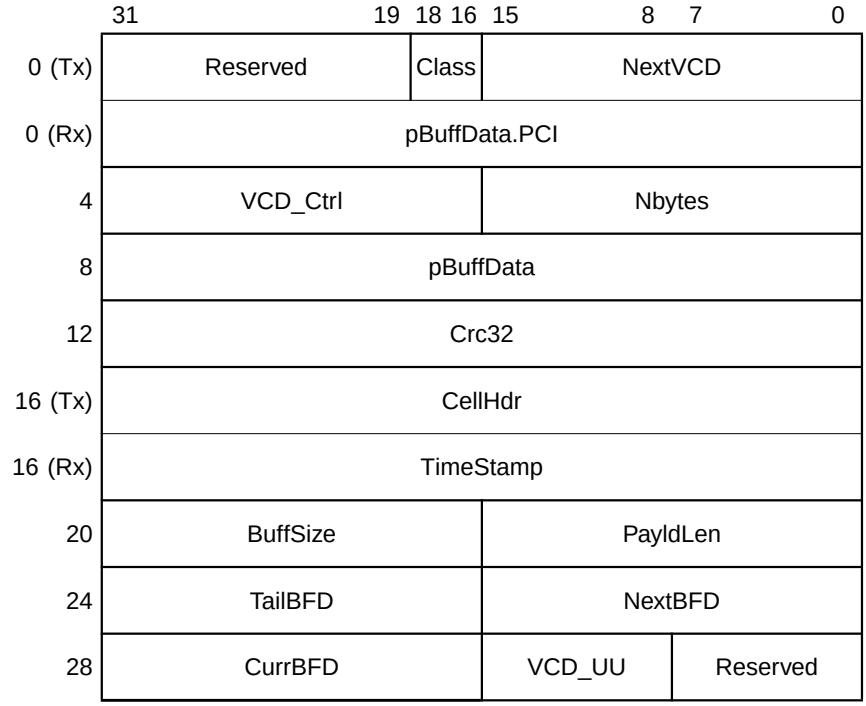
The RxCell Processor reassembles CS-PDU payloads in buffers. When the APU instructs the RxCell Processor to reassemble a cell belonging to a virtual connection, the processor checks if there is enough space in the buffer to store the cell payload. If there is not enough space, the processor returns the current buffer to the Receive Completion Queue and attempts to use a buffer following the current one on the linked list. In case there is no buffer following the current one, the RxCell Processor attaches a buffer from one of the six Small or Large Free Buffer Lists to the VC Descriptor. The free buffer list used is defined by the `VCD_RxCtrl` bits in the `VCD Control` field of the VCD. The VCD structure is described in the next section.

If the APU wants to control the memory management, it can provide buffer numbers for the RxCell Processor in advance. The EDMA always attempts to use existing buffers before linking a new one from a Free Buffer List. This arrangement supports both elaborated buffer management using the APU and a simple, but fast, EDMA (hardware) controlled scheme.

5.2.1 VC Descriptor Structure

As shown in Figure 5.3, the structure of a VC Descriptor differs slightly between the transmit and receive directions. For both transmit and receive, the descriptor occupies eight words and must be aligned on a 32-byte boundary.

Figure 5.3 Virtual Connection Descriptor



Although all the individual fields of a VC Descriptor are described below, the APU does not need to access these fields during normal operation. A VC Descriptor must be initialized by the APU when a connection is established and before the first cell is processed. After initialization, the EDMA manages the descriptor autonomously.

5.2.1.1 VC Descriptor Fields

Table 5.2 provides information about the individual fields in the VC Descriptor. The text that follows the table provides additional information about each field.

The Init column specifies the initialization value the APU must store in the VC Descriptor before the first cell is received or transmitted over the VC. A Yes in the Init column means the APU must place an appropriate value in that field. A 0 in the Init column means the APU must initialize

that field to zero. A dash (–) in the Init column means an initialization value is not relevant (that is, it is don't care).

Table 5.2 VC Descriptor Fields

Field Name	Init	Addr	Dir	Size (Bits)	Description
Reserved	–	–		13	Reserved for future use
Class	Yes	0x01	Tx	3	VC priority class
NextVCD	0	0x02	Tx	16	Next VC in a linked list
pBuffData.PCI	–	0x00	Rx	32	pBuffData in PCI for Rx buffers
VCD_Ctrl	Yes	0x04	Tx/Rx	16	Control bits described in Table 5.3
Nbytes	0	0x06	Tx/Rx	16	Number of bytes left in the buffer
pBuffData	–	0x08	Tx/Rx	32	Pointer to the buffer data
Crc32	–	0x0C	Tx/Rx	32	Partial CRC32 value
CellHdr	Yes	0x10	Tx	32	ATM cell header
TimeStamp	–	0x10	Rx	32	Last time when the connection was serviced
BuffSize	–	0x14	Rx	16	Size of the current buffer
PayldLen	–	0x16	Tx/Rx	16	Total number of bytes in the CS-PDU payload
TailBFD	0	0x18	Tx/Rx	16	Tail of the list of Buffer Descriptors
NextBFD	0	0x1A	Tx/Rx	16	Next Buffer Number
CurrBFD	0	0x1C	Tx/Rx	16	Current Buffer Number
VCD_UU	0	0x1E	Tx/Rx	8	AAL5 User-to-User indication
Reserved	–	–		8	Reserved for future use

Reserved **Tx Word 0 [31:19]**

Do not modify these bits.

Class **VC Priority Class** **Tx Word 0 [18:16]**

This field stores the priority class of the connection. This field is used and maintained by the Scheduler. The EDMA does not access this field. Connections that do not use the Scheduler can use this field for another purpose.

NextVCD **Next VC Descriptor** **Tx Word 0 [15:0]**

The Scheduler uses this field to chain VC Descriptors in a linked list. This field is used and maintained by the Scheduler. The EDMA does not access this field. Connections that do not use the Scheduler can use this field for another purpose.

pBuffData.PCI

Pointer to Buffer Data in PCI **Rx Word 0 [31:0]**

The EDMA uses this field in the Rx VCD to hold the PCI address of the buffer in Buffer Copy mode.

VCD_Ctrl **VC Descriptor Control** **Word 1 [31:16]**

This field stores the control bits of the VC Descriptor. See Section 5.2.1.2, “VC Descriptor Control Field.”

Nbytes **Number of Bytes** **Word 1 [15:0]**

This field represents the number of bytes in the current buffer to be processed.

For the transmit direction, the `Nbytes` field indicates the number of bytes still to be sent from the buffer. The EDMA decrements `Nbytes` and compares it with zero to check when all of the data in a buffer is completely segmented. The EDMA clears that field when all data from the current buffer is sent.

For the receive direction, the `Nbytes` field indicates the number of bytes left free in the buffer. The EDMA compares that value with the number of bytes in the current cell to check if there is enough space in the buffer to store the cell payload. If there is not enough space, the EDMA attempts to use a buffer linked after the current buffer. In case there is no buffer after the current one, the EDMA attaches a buffer from a Free Buffer List.

Free buffers are attached when the Beginning of Message (BOM) cell is received or when the current buffer does not have sufficient space to hold a cell’s payload. Although the BOM cell is not marked in AAL5, the EDMA infers it from the fact that the previous cell was an EOM cell. This information is stored in the VC Descriptor control field (`VCD_ConAct` bit). Buffers from the Small Buffer List are used for BOM cells and Buffers from the Large Buffer List are used for continuation or when the first Buffer List is exhausted.

pBuffData **Pointer to Buffer Data** **Word 2 [31:0]**

This field stores a pointer to the beginning of the current buffer data. The pointer is copied from the Buffer Descriptor when the EDMA accesses a new buffer to avoid accessing Buffer Descriptors on a per cell basis. The pointer is incremented to point to the current location in the buffer. If pBuffData[32] from the `VCD_Ctrl` field is set, then pBuffData is a 32-bit PCI pointer. Otherwise, pBuffData is a 27-bit Secondary Bus pointer.

Crc32 **Partial CRC32** **Word 3 [31:0]**

This field stores the partial CRC32. The EDMA initializes CRC32 at the beginning of a CS-PDU payload and appends (transmit) or verifies (receive) it when processing the last cell of the payload. Note that the `CRC32` field is used to hold additional control fields in the AAL0 mode as described in Section 5.7, “AAL0 Mode Operation.”

CellHdr **Transmit Cell Header** **Tx Word 4 [31:0]**

In transmit VC Descriptors, this word contains the cell header. The EDMA transfers the ATM cell header from the `CellHdr` field in the VC Descriptor to the front of every cell transmitted and sets bit 0 of the `PTI` field in the header of the last cell of the payload when in AAL5 mode to indicate End of Message (EOM).

TimeStamp **Time Stamp Counter Value** **Rx Word 4 [31:0]**

In receive VC Descriptors, this field contains the Time Stamp Counter value at the time the last cell was received successfully. This field is mainly used for PDU aging. The Time Stamp Counter is described in detail in Chapter 8, “Timer Unit.”

BuffSize **Buffer Size** **Word 5 [31:16]**

This field stores the size of the current buffer. To reduce the number of Buffer Descriptor accesses, `BuffSize` is copied along with `NextBFD` from a Buffer Descriptor to the VC Descriptor when the buffer is processed for the first time.

PayldLen **Payload Length** **Word 5 [15:0]**

In AAL5 mode, the EDMA uses this field to compute the CS-PDU payload length. The EDMA adds the lengths of all buffers belonging to the same CS-PDU. In the transmit

direction, PayldLen is placed in the last cell transmitted from the given CS-PDU. In the receive direction, this field is compared with the payload length extracted from the CS-PDU trailer to detect lost cells.

TailBFD **Tail Buffer Number** **Word 6 [31:16]**
This field contains the Buffer Number of the buffer at the tail of the list of buffers attached to the VC Descriptor. The EDMA uses this field when it needs to attach a buffer to the list.

NextBFD **Next Buffer Descriptor** **Word 6 [15:0]**
This field stores the next Buffer Descriptor Number. To reduce the number of Buffer Descriptor accesses, NextBFD is copied along with BuffSize from a Buffer Descriptor to the VC Descriptor when the buffer is processed for the first time.

CurrBFD **Current Buffer Descriptor** **Word 7 [31:16]**
This field stores the current Buffer Descriptor Number.

VCD_UU **AAL5 User-to-User Indicator** **Word 7 [15:8]**
This field is added to the trailer of an AAL5 CS-PDU in the transmit direction and filled with the UUI byte from the CS-PDU trailer in the receive direction.

Reserved **Word 7 [7:0]**
Do not modify these bits.

5.2.1.2 VC Descriptor Control Field

The VC Descriptor Control field, illustrated in Figure 5.4, contains control bits for the VC Descriptor. Table 5.3 provides information about each control bit. The text following the table includes additional information about the control bits.

Figure 5.4 VC Descriptor Control Field

31	30	29	28	27	26	25	24	23 19	18	17	16
VCD_ Buff-Pres	VCD_ ConAct	VCD_BuffCont/ VCD_BuffLarge	VCD_ Buff-Free	pBuff-Data[32]	VCD_ Buff-Done	VCD_ EFCI	VCD_ CLP	VCD_PHY/ VCD_RxCtrl	VCD_ CellHold	VCD_ AAL0	VCD_ ConOpen

Table 5.3 VC Descriptor Control Bits

Bit	Name	Dir	Description	Owner
31	VCD_BuffPres	Tx/Rx	A buffer is attached to the VCD	EDMA
30	VCD_ConAct	Tx/Rx	Connection active status	EDMA
29	VCD_BuffCont	Tx	Current buffer continuation flag	EDMA
29	VCD_BuffLarge	Rx	Buffer retrieved from the Large Free List	EDMA
28	VCD_BuffFree	Tx/Rx	Buffer retrieved from a Free List	EDMA
27	pBuffData[32]	Tx/Rx	Bit to indicate if pBuffData is PCI or Secondary address	EDMA
26	VCD_BuffDone	Tx/Rx	Buffer complete but disposition is dependent on next cell	EDMA
25	VCD_EFCI	Rx/Tx	Cell Header EFCI bit	EDMA
24	VCD_CLP	Rx/Tx	Cell Header CLP bit	EDMA
23:19	VCD_PHY	Tx	Address of the physical device to use	APU
23:19	VCD_RxCtrl	Rx	Rx Control field to hold Free List	EDMA
18	VCD_CellHold	Rx/Tx	Do not trigger ACI for cell transmission/do not return cell to free list	APU
17	VCD_AAL0	Rx/Tx	Type of ATM Adaptation Layer used	APU
16	VCD_ConOpen	Rx/Tx	Connection Open	EDMA

VCD_BuffPres

Buffer Descriptor Present 31

This is a status bit that the EDMA clears when there is no Buffer Descriptor attached to the VC Descriptor. It is set when at least one Buffer Descriptor is attached. It represents the active/inactive state of a VC descriptor.

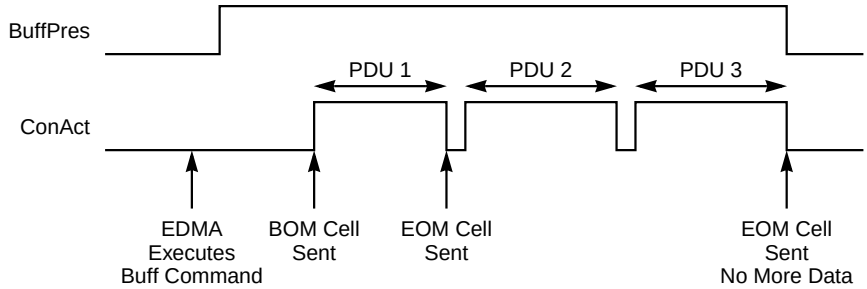
VCD_ConAct Connection Active 30

This is a status bit that indicates when a PDU is under segmentation or reassembly. This bit is set after the BOM cell is processed and cleared after an EOM cell has been processed.

The behavior of `VCD_BuffPres` and `VCD_ConAct` bits in time in the transmit direction is shown in Figure 5.5. The figure assumes that there are no buffers attached to the VC Descriptor initially. When the EDMA executes a `buff`

command, the VCD_BuffPres bit is set. Also, if the EDMA_ConReAct bit (EDMA_Ctrl register, bit 3, page 5-53) is set, the Connection Number is returned to the Completion Queue and the APU may start scheduling the connection for transmission. The VCD_ConAct bit is then set after the BOM cell for this connection is processed. When the last cell from a PDU is sent and there are no more buffers attached to the VC Descriptor, the VCD_BuffPres bit is cleared.

Figure 5.5 BuffPres and ConAct Bits Timing



VCD_BuffCont **29**

Current buffer continuation flag (for EDMA internal usage).

VCD_BuffLarge **29**

Buffer retrieved from a Large Free List.

VCD_BuffFree **28**

Buffer retrieved from a free list. This bit is for EDMA internal usage.

pBuffData[32] **27**

If this bit is set, pBuffData is a PCI pointer. If this bit is cleared, pBuffData points to a Secondary Bus address.

VCD_BuffDone **26**

Buffer complete but disposition is dependent on next cell.

VCD_EFCI and VCD_CLP **[25:24]**

The behavior of these bits depends on the EDMA_OrHdr bit in the EDMA_Ctrl register (page 5-52).

If the EDMA_OrHdr bit is cleared, then the bits 2 and 0 of cell headers are extracted from each received cell and placed in VCD_EFCI and VCD_CLP, respectively. They

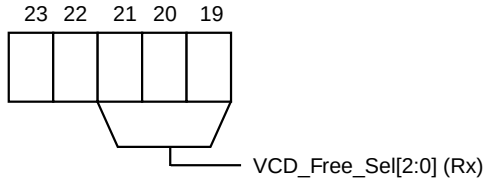
represent the network congestion indication and cell loss priority of the last received cell. In the transmit direction, the bits indicate the current value of `CellHdr[2]` and `CellHdr[0]` which are copied to the free cell in the CBM.

If the `EDMA_OrHdr` bit is set, then the `EFCI` and `CLP` bits of a cell header are controlled and observed from Buffer Descriptors as explained in Section 5.2.2.2, "Buffer Descriptor Control Field."

VCD_PHY **Device Physical Address** **[23:19]**
In the transmit direction, the APU specifies the address of the physical device in this field.

VCD_RxCtrl **Rx Control Field** **[23:19]**
In the receive direction, this field indicates the Free List from which the EDMA attaches buffers to the VCD. See Figure 5.6.

Figure 5.6 VCD_RxCtrl Usage



VCD_CellHold Inhibit Cell **18**
This bit inhibits the automatic cell transmission (Tx) and cell location disposal (Rx).

In the transmit direction, if this bit is cleared, the EDMA automatically puts the cell in the Transmit FIFO when it is completely built in Cell Buffer Memory. If this bit is set, the EDMA does not put the cell in the Transmit FIFO and this task must be performed by the APU.

In the receive direction, when the `VCD_CellHold` bit is cleared, the EDMA automatically returns the cell location to the Cell Buffer Manager when the cell is completely transferred out of Cell Buffer Memory. When this bit is set, the EDMA does not return the cell location to the Cell Buffer Manager.

VCD_AAL0 **Operation Mode** **17**

When set, this bit selects the AAL0 mode of operation; when cleared, it selects the AAL5 mode. EDMA operation in the AAL0 and AAL5 modes is described in Section 5.6, “AAL5 Mode Operation,” and Section 5.7, “AAL0 Mode Operation.”

VCD_ConOpen

Connection Open **16**

The EDMA uses this bit to indicate whether the connection (Tx or Rx) is open or closed. If a TxCell or RxCell command is issued to a closed connection, a BFS_ErrNotOpen status bit is returned to the completion queue. See Figure 5.20 on page 5-37. The APU can close the connection by issuing a TxConClose/ RxConClose command described in Section 5.3.5, “TxConClose/RxConClose Command.”

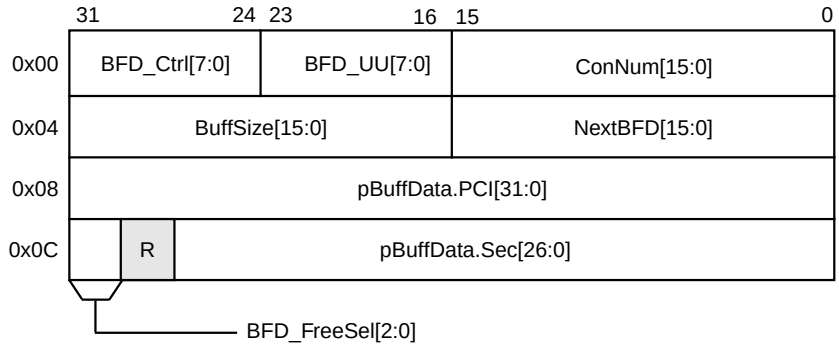
All the EDMA status bits are written after the current cell is processed. For single-cell CS-PDUs, the EDMA sets the VCD_ConAct bit internally when cell processing starts but does not write it to the VC Descriptor until cell processing is finished, at which time the bit is cleared. Therefore, single-cell CS-PDUs do not result in toggling of the VCD_ConAct bit.

The EDMA reads and writes VC Descriptors in words. During most EDMA operations, the Nbytes field and certain bits of the VCD_Ctrl field are changed in EDMA internal registers. The EDMA writes back both VCD_Ctrl and Nbytes fields in a single write operation when the VC is completely serviced. Therefore, to preserve information consistency, the APU must guarantee that the EDMA is not accessing the VC Descriptor that is to be modified. The TxConClose/RxConClose commands may be used by the APU to keep the EDMA from modifying the VC Descriptor.

5.2.2 Buffer Descriptor

A Buffer Descriptor occupies four words in memory and must be aligned on a 16-byte boundary. Figure 5.7 illustrates the layout of the fields in a Buffer Descriptor.

Figure 5.7 Buffer Descriptor



5.2.2.1 Buffer Descriptor Fields

Table 5.4 lists and defines the individual fields contained in a Buffer Descriptor. The text below the table provides additional information about each field.

Table 5.4 Buffer Descriptor Fields

Name	Addr	Size Bits	Description	Initilized
BFD_Ctrl	0x0	8	Buffer Descriptor control bits described in Table 5.5	Yes
BFD_UU	0x1	8	AAL5 User-to-User byte	Yes
ConNum	0x2	16	Connection Number to which buffer belongs	Yes
BuffSize	0x4	16	Number of bytes in the buffer	Yes
NextBFD	0x6	16	The next Buffer Descriptor in the list	Yes

Table 5.4 Buffer Descriptor Fields (Cont.)

Name	Addr	Size Bits	Description	Initilized
pBuffData.PCI	0x8	32	PCI pointer to the payload	Yes
BFD_FreeSel	0xC	3	Free List select	Yes
pBuffData.Sec	0xC	27	Secondary Bus pointer to payload	Yes

BFD_Ctrl **Buffer Descriptor Control** **Word 0 [31:24]**
This field stores the control bits of a Buffer Descriptor.

BFD_UU **AAL5 User-to-User Byte** **Word 0 [23:16]**
Data byte reserved for user.

ConNum **Connection Number** **Word 0 [15:0]**
This field contains the Connection Number to which the Buffer Descriptor is attached. In the receive direction, the EDMA builds this field when the buffer data is completed and before the buffer number is returned to the EDMA Completion Queue. In the transmit direction, the host or the APU must set the `ConNum` field and the EDMA uses it to determine the VC Descriptor to which it should attach the buffer when executing a `buff` command.

BuffSize **Buffer Size** **Word 1 [31:16]**
This field indicates the number of bytes in the buffer. In the transmit direction it is set by the host or the APU to specify how many bytes of data are present in the buffer. In the receive direction and for the buffers that are attached in advance, the `BuffSize` field should be set by the host or the APU to specify the size of the buffer. This field is ignored for buffers taken from a free list. Instead, the EDMA uses values from the EDMA_SBuffSize or EDMA_LBuffSize registers (page 5-49) depending on whether the buffer was retrieved from the Small or from the Large Free Buffer List. The number of bytes received is returned in the `BuffSize` field when the EDMA has completed processing the receive buffer.

NextBFD **Next Buffer Descriptor** **Word 1 [15:0]**
This field contains the number of the next buffer on a linked list or zero at the end of the list. The Buff Processor maintains this field for Tx Buffer Descriptors.

pBuffData.PCI

PCI Buffer Data Pointer **Word 2 [31:0]**

Either the APU or the host stores a buffer payload pointer to PCI address space in this field. The EDMA uses the pointer to find the beginning of the buffer payload. This field is never modified by the EDMA. This field, along with the `pBuffData.Sec` field, determines if the buffer is in the Buffer Copy or Cell mode.

BFD_FreeSel **Free List Select** **Word 3 [31:29]**

The APU uses this field to indicate to which Free List (0–5) the BFD belongs.

R **Reserved** **Word 3 [28:27]**

Do not change these bits.

pBuffData.Sec

Secondary Bus Buffer Data Pointer **Word 3 [26:0]**

Either the APU or the host stores a payload pointer to Secondary Bus address space in this field. The EDMA uses the pointer to find the beginning of the buffer payload. This field is never modified by the EDMA. This field, along with the `pBuffData.PCI` field, determines if the buffer is in the Buffer Copy or Cell mode.

Note: The BFD is in the “packet mode” if both the `pBuffData.PCI` and `pBuffData.Sec` fields are nonzero and the data is copied from the PCI to the Secondary memory at the `Buff` command. In the receive direction, the `Buff` command can be used to preattach the “packet mode” BFD to the VCD. If either `pBuffData.PCI` or `pBuffData.Sec` is 0, then the buffer is in Cell mode and the buffer is not copied.

5.2.2.2 Buffer Descriptor Control Field

The Buffer Descriptor Control field, illustrated in Figure 5.8, stores control bits for the Buffer Descriptor. Table 5.5 provides information about each control bit. The text following the table includes additional information about the control bits.

Figure 5.8 Buffer Descriptor Control Field

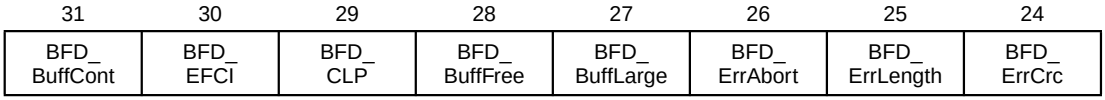


Table 5.5 Buffer Descriptor Control Bits

Bit	Name	Dir	Description	Init
31	BFD_BuffCont	Rx/Tx	Buffer continuation	Yes
30	BFD_EFCI	Rx/Tx	EFCI bit of cell header	Yes
29	BFD_CLP	Rx/Tx	CLP bit of cell header	Yes
28	BFD_BuffFree	Rx	Buffer retrieved from a Free List	0
27	BFD_BuffLarge	Rx	Buffer from Large Free List	0
26	BFD_ErrAbort	Rx	Zero payload length in cell	0
25	BFD_ErrLength	Rx	Lost or misinserted cell	0
24	BFD_ErrCrc	Rx	Crc32 error	0

BFD_BuffCont

Buffer Continued **31**

The APU or the host sets this bit in the transmit direction and the EDMA sets it in the receive direction. When this bit is set, it indicates that the CS-PDU payload is continued in the buffer following the current one and pointed to by `NextBFD`.

BFD_EFCI and BFD_CLP **[30:29]**

When the `EDMA_OrHdr` bit in the EDMA_Ctrl register (page 5-52) is set, the `BFD_EFCI` and `BFD_CLP` bits generate or extract the corresponding bits of a cell header on a per CS-PDU basis.

In the transmit direction, the `BFD_EFCI` and `BFD_CLP` bits of the first buffer of a CS-PDU are logically ORed with corresponding bits of the `VCD_CellHdr` to compute the cell header bits of all cells belonging to that CS-PDU:

```
CellHeader[0] = BFD_CLP | VCD_CellHdr[0]
CellHeader[2] = BFD_EFCI | VCD_CellHdr[2]
```

In the receive direction, the BFD_EFCI and BFD_CLP bits are set for the last buffer of a CS-PDU if any cell belonging to the CS-PDU has the corresponding cell header bit set. Note that the EDMA RxCell Processor does not read the Cell Buffer Memory to obtain cell headers but rather uses bits provided by the APU with the RxCell command (see Figure 5.10).

When the EDMA_OrHdr bit is cleared, cell header bits 2 and 0 are controlled and observed using the VCD_CLP and VCD_EFCI bits. The BFD_CLP and BFD_EFCI bits are discarded in the transmit direction and never set in the receive direction.

BFD_BuffFree

Buffer Free **28**

The EDMA sets this bit in the receive direction when the buffer is retrieved from a Free Buffer List.

BFD_BuffLarge

Buffer Large **27**

This bit is set for buffers from the Large Free Buffer list and cleared for buffers from the Small Free Buffer List.

BFD_ErrAbort

Zero Payload **26**

This bit is set in the receive direction when the payload field of an AAL5 EOM cell holds a zero value.

BFD_ErrLength

Payload Length Error **25**

The EDMA sets this bit if an incorrect payload length is detected. The EDMA detects lost or misinserted cells by comparing the accumulated payload length, LenVCD, with the payload length, LenCell, extracted from the CS-PDU trailer. The accumulated payload length is the number of received cells since the last EOM cell multiplied by 48 (including the current EOM cell). The BFD_ErrLength bit is set if the following relation is not detected (receive direction only):

$$\text{LenCell} + 8 \leq \text{LenVCD} < \text{LenCell} + 56$$

BFD_ErrCrc **CRC32 Error** **24**
The EDMA sets this bit in the receive direction when the computed CRC32 does not match the one extracted from an EOM cell.

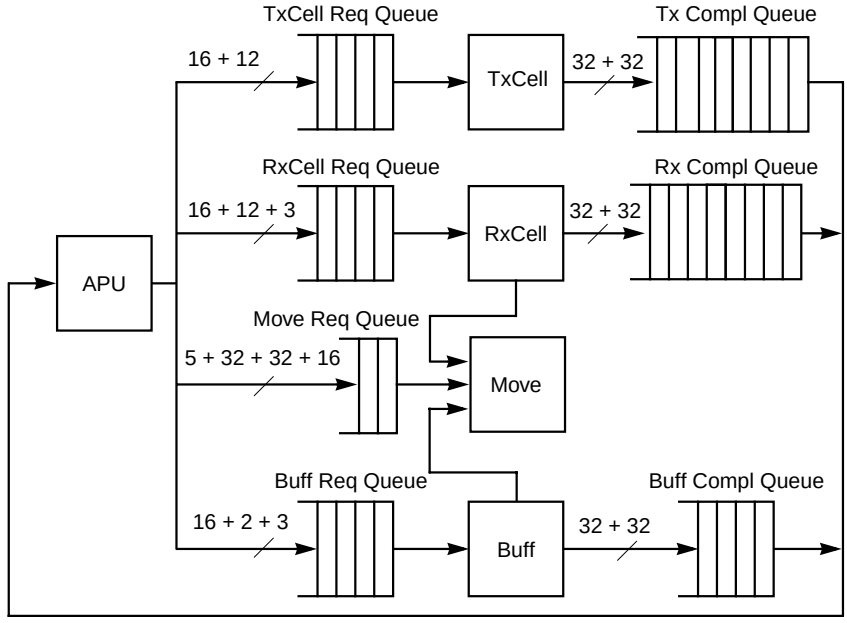
5.3 EDMA Commands

Memory mapped registers serve as the interface between the EDMA and the APU. The interface handles the following types of requests:

- Transmit cells to or receive cells from a virtual connection
- Attach a buffer to a VC Descriptor
- Transfer data between source and destination locations

These requests are made by writing commands to the appropriate memory mapped registers. Each register has an associated request queue. After a command is written to a register, the command is then written to the appropriate request queue (if that request queue is not full). Figure 5.9 illustrates the request queues.

Figure 5.9 EDMA Request & Completion Queues



The status of command execution can be checked by reading the EDMA_Status register (page 5-33).

If the APU writes to a command register and the associated request queue is full, the APU is stalled until the EDMA processor retrieves the command and renders the request queue not full. This situation needs to be avoided since it can create a deadlock. For example, a deadlock occurs when the APU stalls as the result of writing to a full request queue at the same time the EDMA processor stalls as the result of writing to a full completion queue. This situation leads to a Watchdog Timer Bus Error which is usually a nonrecoverable system exception.

The feedback path from the EDMA processors to the APU uses three EDMA completion queues. The TxCell and RxCell Completion queues are eight entries deep while the Buff Completion queue is four entries deep. The EDMA mainly uses the completion queues to send completed Buffer Numbers back to the APU. However, the queues are also used to signal exception and error conditions. Since an EDMA processor is stalled when it attempts to write to a full completion queue, the APU should empty the queues at a sufficient rate to avoid the problem (by reading the EDMA_TxCompl, EDMA_RxCompl, or EDMA_BuffCompl registers, page 5-49).

See Table 5.7 through Table 5.9 on page 5-42 for lists of the completion queue messages and their descriptions.

In addition, the Buff Processor is capable of informing the APU that a Buffer Descriptor has been attached to a VC Descriptor that previously did not have any buffers, possibly requiring connection rescheduling.

The RxCell and Buff Processors can issue commands to the Move Processor when Buffer Copy mode is selected (see Section 5.4.3, “Buffer Payload.”)

The Move Processor gives priority to the EDMA (`RxCell` or `Buff` commands) over the APU. Once a move begins, however, it is not interrupted for a higher priority command. The EDMA `RxCell` command has priority over the `Buff` command. Again, lower priority commands in process are not interrupted by higher priority commands.

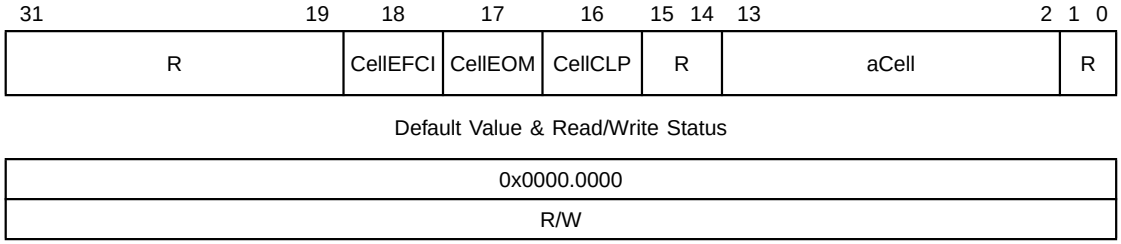
Note: Transmit buffer copies are initiated by the Buff Processor.

For information about other registers that are used to control EDMA operations, see Section 5.5, “Register Descriptions.”

5.3.1 RxCell Command

The RxCell command is used to put a cell service request in the RxCell Request Queue. The APU has to write the desired connection number to the EDMA_RxConNum register (page 5-49) and write the cell address, which is concatenated with the three LSBs of the cell header, to the EDMA_RxCell register (Figure 5.10). The second write operation transfers the contents of both registers into the RxCell Request Queue (if it is not full).

Figure 5.10 EDMA_RxCell Register Format



The CellEFCI, CellEOM, and CellCLP bits correspond to bits 2–0 of a cell header in AAL5 mode. The EDMA_RxCell register is located at address 0xB800.0048.

R **Reserved** **[31:19]**
Not used by the L64364.

CellEFCI **Cell EFCI** **18**
When set, this bit causes the EDMA to extract the corresponding bits from the cell header and store them first in a VC Descriptor (VCD_EFCI and VCD_CLP bits) and, when a buffer is completed, in a Buffer Descriptor (BFD_EFCI and BFD_CLP bits). The EDMA stores either the values from the last cell command or a logical OR of all bits from the CS-PDU. Which value the EDMA stores depends on the EDMA_OrHdr bit located in the EDMA_Ctrl register (page 5-52).

CellEOM **Cell End of Message** **17**
When set, this bit indicates that the current cell is the last cell of a CS-PDU in AAL5 mode. The AAL0 mode uses the

CDS_EOM field of the Cell Descriptor (see Section 5.7, “AAL0 Mode Operation,” and Section 6.3, “Cell Descriptor.”)

CellCLP **Cell CLP** **16**
See the CellEFCI description above.

R **Reserved** **[15:14]**
Not used by the L64364. See the aCell field description following.

aCell [11:0] **Cell Address** **[13:2]**
This field represents the cell address in the Cell Buffer Memory. Since cells must be aligned on a word boundary in the Cell Buffer Memory and the maximum cell buffer size is 4 Kbytes, bits [15:14] and [1:0] of the cell address are discarded.

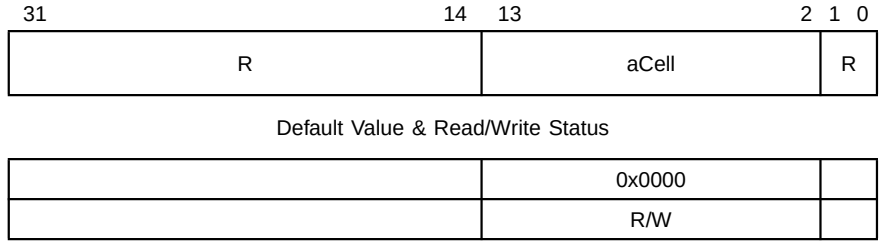
R **Reserved** **[1:0]**
Not used by the L64364. See the preceding aCell field description.

When the RxCell Processor completes a buffer, either because the buffer storage has been exhausted or because an EOM cell has been received, it places the Buffer Number in the Receive Completion Queue. It may also optionally copy the corresponding Buffer Descriptor and issue a command to the Move Processor to copy the buffer contents. These optional behaviors are controlled by the EDMA_RxBFD_Copy bit in the EDMA_Ctrl register (page 5-52) and the pBuffData.Sec and pBuffData.PCI fields of the BFD, respectively. For additional information, see Section 5.2.2, “Buffer Descriptor.”

5.3.2 TxCell Command

The TxCell command is used to put a cell service request in the TxCell Request Queue. The APU has to write the desired Connection Number to the EDMA_TxConNum register (page 5-49) and the cell address to the EDMA_TxCell register (Figure 5.11). The second write operation transfers the contents of both registers into the TxCell Request Queue (if it is not full). The EDMA _TxCell register is located at address 0xB800.0008.

Figure 5.11 EDMA_TxCell Register Format



R **Reserved** **[31:14]**
Not used by the L64364.

aCell[11:0] **Cell Address** **[13:2]**
This field represents the cell address in the Cell Buffer Memory. Cells must be aligned on a word boundary in the Cell Buffer Memory and the maximum cell buffer size is 4 Kbytes, so only 12 bits are needed.

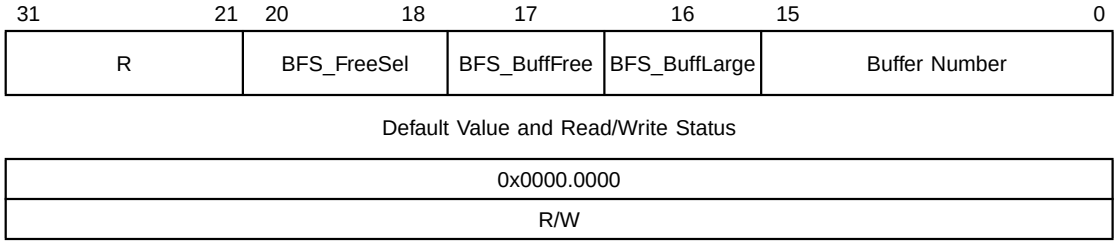
R **Reserved** **[1:0]**
Not used by the L64364.

When the TxCell Processor completes a buffer after all the buffer payload has been sent out, it places the Buffer Number in the Transmit Completion Queue.

5.3.3 Buff Command

The `buff` command is used to attach a Buffer Descriptor to a VC Descriptor for segmentation or reassembly. The command is placed in the Buff Request Queue by writing the Buffer Number and five control bits into the EDMA_Buff register. The register is located at address 0xB800.0080. The format of the register is shown in Figure 5.12.

Figure 5.12 EDMA_Buff Register Format



R **Reserved** **[31:21]**
Not used in the L64364.

BFS_FreeSel[3:0]
Free List Select **[20:18]**
These bits select one of the six (0–5) Free Lists to which the buffer should be returned. If BFS_FreeSel is 6 or 7, the buffer number is returned in the Buff Completion queue.

BFS_BuffFree
Buffer Free **17**
When this bit is set, the EDMA attaches the Buffer Descriptor to the Large (BFS_BuffLarge set) or Small (BFS_BuffLarge cleared) Free Buffer List.

BFS_BuffLarge
Buffer Large **16**
When this bit and the BFS_BuffFree bit are set, the EDMA attaches the Buffer Descriptor to a Large Free Buffer List. When this bit is cleared and the BFS_BuffFree bit is set, the EDMA attaches the Buffer Descriptor to a Small Free Buffer List. If the BFS_BuffLarge bit is set and the BFS_BuffFree bit is cleared, then the contents of the buffer are not copied from the pBuffData.PCI to the pBuffData.Sec when a “packet mode” buffer is attached to the VC descriptor and both the fields are nonzero (see Section 5.2.2.1, “Buffer Descriptor Fields”). This can be used to preattach buffers to the receive VC Descriptor.

Buffer Number[15:0] **[15:0]**
16-bit number assigned to a Buffer Descriptor location at initialization.

When the EDMA retrieves the buff command with a Buffer Number from the request queue, it reads a Connection Number from the Buffer Descriptor and attaches the Buffer Descriptor at the end of the Buffer Descriptor list for the connection. Since the EDMA disregards the NextBFD field of the Buffer Descriptor only one Buffer Descriptor may be attached with a single buff command.

The command may be used for both transmit and receive virtual connections. For the transmit side, the linked list of Buffer Descriptors is used by the EDMA to segment the buffer data into cells. For the receive direction, the command is optional since the EDMA uses a storage area from a Free Buffer List if there are no Buffer Descriptors available.

When the Buff Processor attaches a buffer, it also optionally may copy the corresponding Buffer Descriptor and issue a command to the Move Processor to copy the buffer contents. These optional behaviors are controlled by the EDMA_TxBFD_Copy and EDMA_RxBFD_Copy bits in the EDMA_Ctrl register (page 5-52), and the nonzero pBuffData.PCI and pBuffData.Sec fields in the BFD, respectively. The behaviors are further described in Section 5.2.2, “Buffer Descriptor.” The encoding of BFS_BuffFree, BFS_BuffLarge, and BFS_FreeSel is summarized as follows.

Table 5.6 BFS_BuffFree, BFS_BuffLarge, and BFS_FreeSel Encoding

BFS_ BuffFree	BFS_ BuffLarge	BFS_ FreeSel	Operation
0	0	0	Attach a BFD to VCD (copy BFD/buffer as specified by EDMA_TxBFD_Copy and BFD pBuffData pointers). ConReact message if applicable.
0	1	0	Attach a BFD to VCD (no buffer copy, copy BFD as specified by EDMA_RxBFD_Copy). No ConReact messages.
1	0	0–5	Return a BFD to small free list.
1	1	0–5	Return a BFD to large free list.

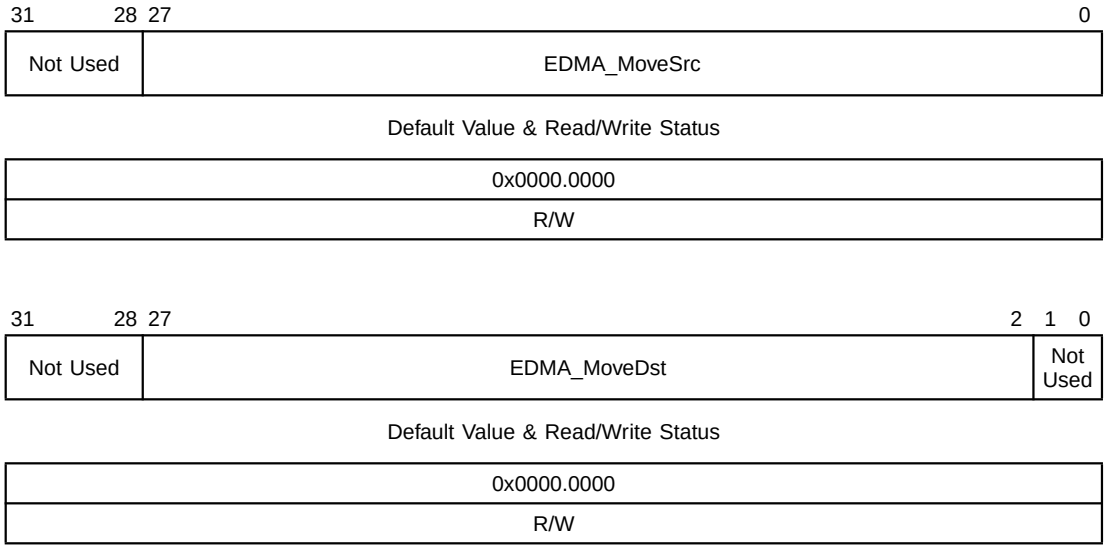
The Buff Processor places the Connection Number in the EDMA_BuffCompl register to inform the APU that the connection might require rescheduling if:

- the VC Descriptor has no buffers attached when the command is executed,
- the VCD_BuffPres bit toggles from zero to one as a result of command execution, and
- the EDMA_ConReAct bit is set.

5.3.4 Move Command

The move command transfers a block of data between the Secondary Bus and the PCI Bus. To issue the move command, first write source and destination addresses to the EDMA_MoveSrc and EDMA_MoveDst registers, respectively (see Figure 5.13). Then write the number of bytes to be transferred to the EDMA_MoveCount or EDMA_MoveCount2 register (see page 5-29 or page 5-31). The move command is placed in the Move Request Queue when a nonzero byte count is written to the EDMA_MoveCount or EDMA_MoveCount2 register. The EDMA_MoveSrc register is located at address 0xB800.00A0 and the EDMA_MoveDst register is located at address 0xB800.00A4.

Figure 5.13 EDMA_MoveSrc and EDMA_MoveDst Register Format



The Move Processor cannot perform byte alignment. The two LSBs of the EDMA_MoveSrc register are used for the two LSBs of both the source and destination address. The two LSBs from the EDMA_MoveDst register are ignored.

When initiating the move command using the EDMA_MoveCount register, the four MSBs of the EDMA_MoveSrc and EDMA_MoveDst registers are not used, and bit 27 of the EDMA_MoveSrc is used to determine the direction of the transfer. If EDMA_MoveSrc[27] = 0, then the transfer is from the Secondary Bus to the PCI Bus. If EDMA_MoveSrc[27] = 1, the transfer is from PCI Bus to Secondary Bus. The 32-bit PCI address is formed by concatenating the 5-bit APU_PriMSB field of the APU_AddrMap register and bits [26:0] of the EDMA_MoveSrc or EDMA_MoveDst register based on the direction of transfer. This form of the move command remains compatible with what was implemented in ATMizer II (L64363).

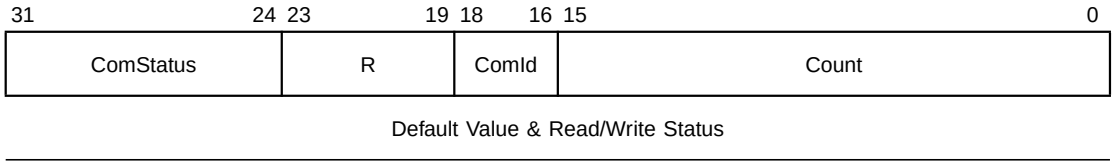
PCI Address = {APU_PriMSB, EDMA_MoveSrc/Dst[26:2], EDMA_MoveSrc[1:0]}

When initiating the move command using the EDMA_MoveCount2 register, the EDMA_MoveSrc and EDMA_MoveDst registers contain 32-bit source and destination addresses. Bit 20 (Sec2PCI) of the EDMA_MoveCount2 register determines the direction of transfer. If Sec2PCI = 0, then the transfer is from the PCI Bus to the Secondary Bus. If Sec2PCI = 1, then the transfer is from the Secondary Bus to the PCI Bus.

Note: The move command cannot be used to transfer a block of data when both the source and destination addresses are located in the same port.

Figure 5.14 illustrates the format of the EDMA_MoveCount register. It is located at address 0xB800.00A8.

Figure 5.14 EDMA_MoveCount Register



31 24	23 19	18 16	15 0
0x00		0x0	0x0000
R/W		Write Only	R/W

ComStatus[7:0] **Command Status** **[31:24]**
See ComId below.

R **Reserved** **[23:19]**
Not used in the L64364.

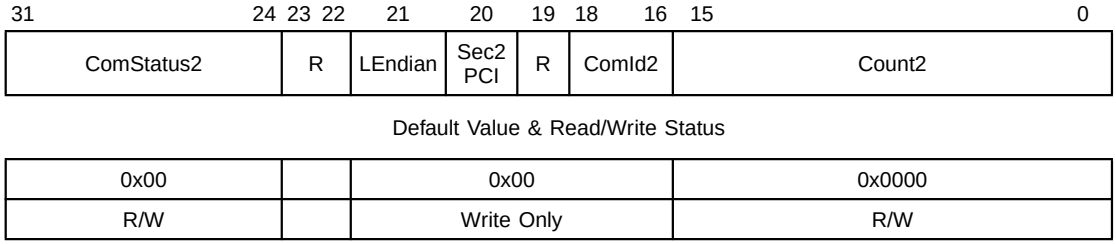
ComId[2:0] **Command ID** **[18:16]**
The ComId and ComStatus fields work together to provide the APU a mechanism for monitoring the status of move commands. When the move command is issued (writing a nonzero value to the Count field), the ComId field is passed to the Move Request Queue. On completion of the move command, the Move Processor sets the corresponding bit in the ComStatus field to indicate that the move command has completed. The APU is informed about the completed command by polling the ComStatus field or by enabling the IntMove_Compl interrupt (see Section 4.8.2, “External Vectored Interrupt Sources”). A ComStatus bit is cleared by writing a “1” to the desired bit.

Note: If a move command is issued with the same ComId as an active or pending move command, the APU may only see a single completion indication for multiple move commands.

Count[15:0] **Move Count** **[15:0]**
This field specifies the number of bytes to be moved.

The EDMA_MoveCount2 register, shown in Figure 5.15, supports 32-bit addressing on the PCI Bus and little and big endian addressing. A 32-bit move is performed by writing 32-bit addresses into the EDMA_MoveSrc and EDMA_MoveDst registers and then writing to the EDMA_MoveCount2 register. The EDMA_MoveCount2 register is located at address 0xB800.00AC.

Figure 5.15 EDMA_MoveCount2 Register



ComStatus2[7:0] **Command Status 2** **[31:24]**
See ComId2 following.

R **Reserved** **[23:22]**
Not used in the L64364.

LEndian **Little Endian** **21**
If LEndian = 0, the Move Processor reads from and writes to both PCI and SB memories as big endian memories, i.e., byte 0 at bits [31:24] and byte 3 at bits [7:0]. If LEndian = 1, the Move Processor reads from and writes to both PCI and SB memories as little endian memories, i.e., byte 3 at bits [31:24] and byte 0 at bits [7:0].

Sec2PCI **Secondary/Bus to PCI Bus Move** **20**
When this bit is set, the move is from the Secondary Memory to the PCI Memory. When it is cleared, the move is in the opposite direction.

R **Reserved** **19**
Not used in the L64364.

ComId2[2:0] **Command ID 2** **[18:16]**
The ComId2 and ComStatus2 fields work together to provide the APU a mechanism for monitoring the status of move commands. When the move command is issued (writing a nonzero value to the Count2 field), the ComId2 field is passed to the Move Request Queue. On completion of the move command, the Move Processor sets the corresponding bit in the ComStatus2 field to indicate that the move command has completed. The APU is informed about the completed command by polling the ComStatus2 field or by enabling the

IntMove_Compl interrupt (see Section 4.8.2, “External Vectored Interrupt Sources”). A ComStatus2 bit is cleared by writing a “1” to the desired bit.

Note: If a move command is issued with the same ComId2 as an active or pending move command, the APU may only see a single completion indication for multiple move commands.

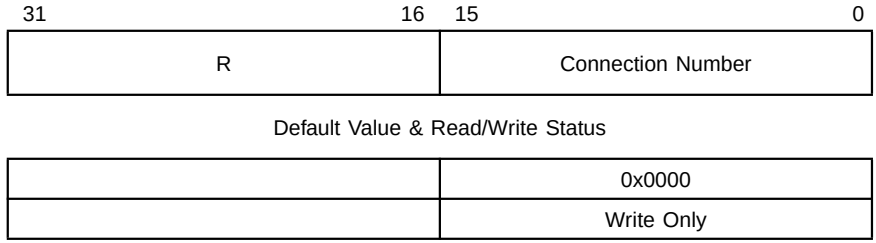
Count2[15:0] Move Byte Count 2 [15:0]
This field specifies the number of bytes to be moved.

The Move Processor can also receive commands internally from the Buff or RxCell Processors that have higher priority than APU commands. However, in all cases, a command that is in progress will always be completed before accepting a new command from the Request Queue or from the other processors.

5.3.5 TxConClose/RxConClose Command

The ConClose commands are used by the APU to clear the VCD_ConOpen bits for the connections that are specified. The commands are executed by writing the EDMA_TxConClose or EDMA_RxConClose register with the connection number the APU would like to access. When the commands are completed, there are ConClose messages in the appropriate completion queues. See Section 5.3.7, “Buffer Completion,” page 5-35 for more information.

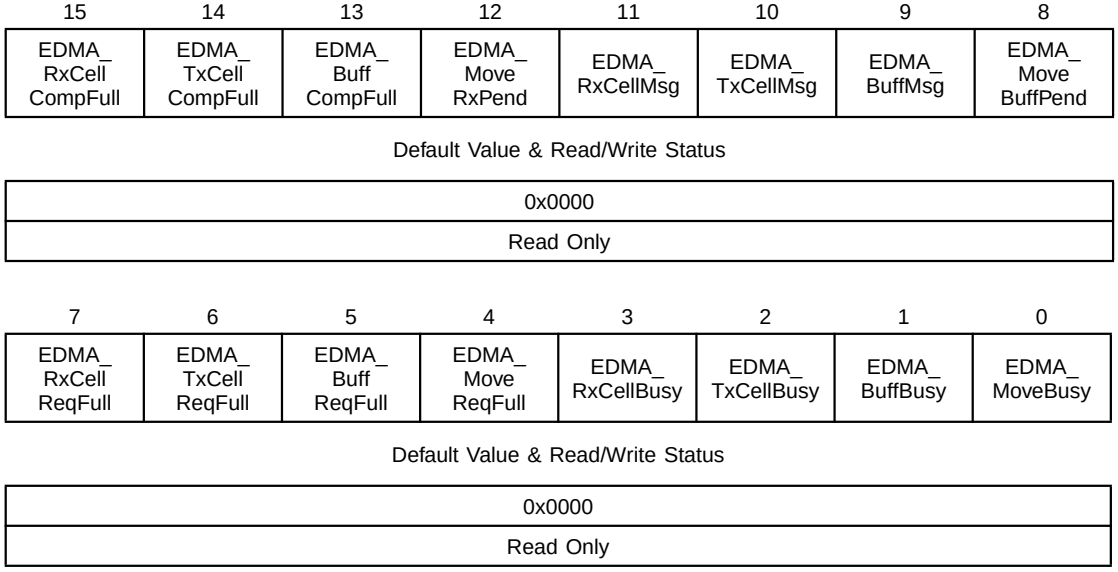
Figure 5.16 Tx/RxConClose Command Format



5.3.6 Checking Status

The APU can check the status of all EDMA processors and queues by reading the EDMA_Status register. The status bits are set when the described conditions occur and are cleared when the conditions are cleared. The register is located at address 0xB800.00C4.

Figure 5.17 EDMA_Status Register



EDMA_RxCellCompFull

RxCell Completion Queue Full **15**

This bit is set if the RxCell Completion Queue is full.

EDMA_TxCellCompFull

TxCell Completion Queue Full **14**

This bit is set if the TxCell Completion Queue is full.

EDMA_BuffCompFull

Buff Completion Queue Full **13**

This bit is set if the Buff Completion Queue is full.

EDMA_MoveRxPend

Rx Move Request Pending **12**

This bit is set when the RxCell Processor has an internally generated request for the Move Processor pending.

EDMA_RxCellMsg

RxCell Completion Queue Message **11**

This bit is set if there is a message in the RxCell Completion Queue.

EDMA_TxCellMsg

TxCell Completion Queue Message **10**

This bit is set if there is a message in the TxCell Completion Queue.

EDMA_BuffMsg

Buff Completion Queue Message **9**

This bit is set if there is a message in the Buff Completion Queue.

EDMA_MoveBuffPend

Buff Move Request Pending **8**

This bit is set when the Buff Processor has an internally generated request for the Move Processor pending.

EDMA_RxCellReqFull

RxCell Request Queue Full **7**

This bit is set if the RxCell Request Queue is full. The APU should avoid writing to the EDMA_RxCell command register, when the `EDMA_RxCellReqFull` bit is set because this will stall the APU.

EDMA_TxCellReqFull

TxCell Request Queue Full **6**

This bit is set if the TxCell Request Queue is full. The APU should avoid writing to the EDMA_TxCell command register, when the `EDMA_TxCellReqFull` bit is set because this will stall the APU.

EDMA_BuffReqFull

Buff Request Queue Full **5**

This bit is set if the Buff Request Queue is full. The APU should avoid writing to the EDMA_Buff command register, when the `EDMA_BuffReqFull` bit is set because this will stall the APU.

EDMA_MoveReqFull

Move Request Queue Full **4**

This bit is set if the Move Request Queue is full. The APU should avoid writing to the EDMA_MoveCount and EDMA_MoveCount2 command registers, when the `EDMA_MoveReqFull` bit is set because this will stall the APU.

EDMA_RxCellBusy

EDMA RxCell Processor Busy **3**

This bit is set when the RxCell Processor is executing an `RxCell` command.

EDMA_TxCellBusy

EDMA TxCell Processor Busy **2**

This bit is set when the TxCell Processor is executing a `TxCell` command.

EDMA_BuffBusy

EDMA Buff Processor Busy **1**

This bit is set when the Buff Processor is executing a `Buff` command.

EDMA_MoveBusy

EDMA Move Processor Busy **0**

This bit is set when the Move Processor is executing a `Move` command.

In addition, the APU may check the current active Connection Number by reading one of the EDMA_XxxxConAct registers (page 5-49).

5.3.7 Buffer Completion

The TxCompl, RxCompl, and BuffCompl commands retrieve completed Buffer Numbers from the EDMA Transmit, Receive, and Buff Completion Queues. A completion queue is a 64-bit wide register that is accessed as a pair of 32-bit registers. The first completion queue is known as the primary completion queue and the second completion queue is referred to as the auxiliary completion queue. The commands are executed by reading the EDMA_TxCompl, EDMA_RxCompl, or EDMA_BuffCompl registers (page 5-49).

A transmit buffer is completed when all data from the buffer is segmented into cells and sent to the ATM Cell Interface (ACI). A receive buffer is completed when either the whole CS-PDU payload is received or the buffer storage is exhausted and a new buffer will be used to reassemble the payload. A `Buff` command is completed when a buffer is attached to a VC Descriptor or a buffer is returned to the Free List.

The EDMA has three completion queues known as EDMA_TxCompl, EDMA_RxCompl, and EDMA_BuffCompl for the Tx processor, Rx processor, and Buff processor, respectively. The primary completion

queue must always be read first. The contents of the auxiliary completion queue are valid after reading the primary completion queue. The auxiliary completion queue contents are accessed by reading the EDMA_TxComplB, EDMA_RxComplB, and EDMA_BuffComplB registers. You may choose not to read the auxiliary completion queue and proceed to read the next entry in the primary completion queue.

The `TxCompl`, `RxCompl`, or `BuffCompl` command returns the value zero when the EDMA Transmit, Receive, or Buff Completion Queue is empty. The EDMA processor is stalled when it attempts to write to a full completion queue. The APU should, therefore, empty the completion queues at a rate sufficient to prevent this problem.

Figure 5.18 shows the contents of the primary completion queue when it holds a buffer number or a connection number.

Figure 5.18 Primary Completion Queue

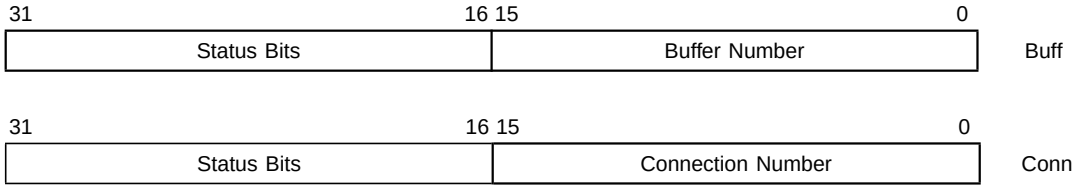


Figure 5.19 shows the contents of the auxiliary completion queue when it holds a connection number, a buffer number, or a cell address.

Figure 5.19 Auxiliary Completion Queue

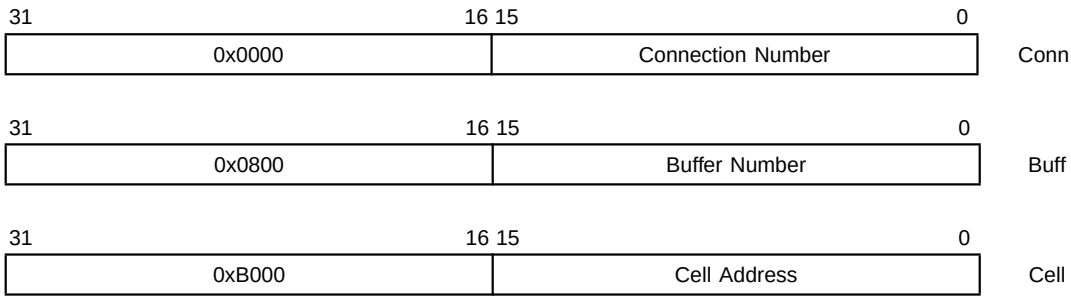
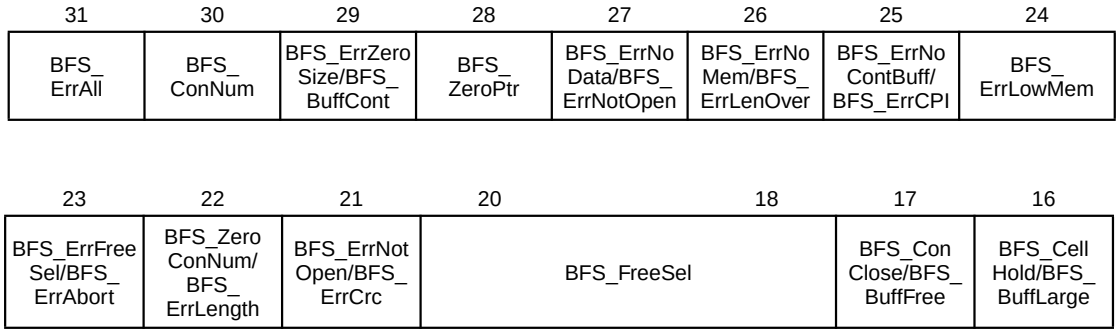


Figure 5.20 shows the status bits returned in the primary completion queue.

Note: Status bit 30 determines the meaning of the dual-function bits. When bit 30 is set, the first definitions of the dual-function bits apply. When it is cleared, the second functions apply.

Figure 5.20 Buffer Status Bits



BFS_ErrAll **All Errors** **31**

This bit is set when any error bit is set in bit positions [29:21].

BFS_ConNum

Connection Number **30**

This bit is set when the completion queue contains a Connection Number and is cleared when the completion queue contains a Buffer Number.

BFS_ErrZeroSize

Zero Size Error (BFS_ConnNum Set) **29**

This bit is set when a `Buff` command for a BFD within the EDMA_Buff register with `BuffSize` equal to zero is issued and `BFS_BuffFree` in the EDMA_Buff register is cleared.

BFS_BuffCont

Buffer Continued (BFS_ConnNum Cleared) **29**

This bit is copied from the Buffer Descriptor. In the Buffer Descriptor, the APU or the host sets this bit in the transmit direction and the EDMA sets it in the receive direction. When this bit is set, it indicates that the CS-PDU payload is continued in the buffer following the current one and pointed to by the `NextBFD` field in the BFD.

BFS_ErrZeroPtr

BFD has Zero Pointers **28**

This bit is set when the EDMA receives a `Buff` command for a BFD with zero pointers or an `RxCell` command which attaches a BFD with zero pointers from the Free List.

BFS_ErrNoData

No Data Error (BFS_ConnNum Set) **27**

This bit is set when the EDMA receives a `TxCell` command for a transmit VC but there is no buffer attached to the VC Descriptor. The Connection Number is placed in the completion queue.

BFS_ErrNotOpen

VCD Closed (BFS_ConnNum Cleared) **27**

This bit is set when the EDMA receives a `Buff` command for a VC which has the `VCD_ConOpen` bit cleared.

BFS_ErrNoMem

Both Free Buffer Lists Empty (BFS_ConnNum Set) **26**

If the RxCell Processor needs a new buffer, it first checks whether there is one attached to the VC Descriptor. If there is not, it takes a buffer from a Free Buffer List. When the RxCell Processor tries to use a buffer from a Small Free Buffer List and the list is empty, it attempts to use the Large Free Buffer List. Similarly, when the processor tries to use a buffer from the Large Free Buffer List and the list is empty, it attempts to use a buffer from the Small Free Buffer list.

This bit is set when the processor attempts to retrieve a buffer from both free lists and both lists are empty. The Connection Number is returned to the completion queue and VCD_ConOpen is cleared (the connection is closed) in the VCD. The connection is closed for recovery purposes as multiple commands for the same VCD may need to be recovered from the Rx request queue before the buffer problem is fixed.

If you do not care about recovering the PDU, then you can drop the cell that originated from the nomem message by returning it to the free list and reopen the connection by setting the VDC_ConOpen bit.

If you do care about recovering the PDU, then you need to stop issuing Rx cell commands and wait for the Rx request queue to empty. The request queue is empty when EDMA_RxCellBusy in the EDMA_Status register is cleared. If there were more cell commands relating to the connection that received the nomem message, then these cells are returned with an Rx completion message of type ErrNotOpen. The APU needs to keep track of the cells that are returned so that they can be reissued after a new data buffer is attached to the VCD and the connection has been reopened.

BFS_ErrLenOver

Payload Length Overflow Error (BFS_ConnNum Cleared) 26

This bit is set when the AAL5 Payload is greater than 64 Kbytes. The VCD_ConOpen bit in the VCD_CTRL register is cleared by the EDMA TxCell Processor.

BFS_ErrNoContBuff

No Continued Buffer Error (BFS_ConnNum Set) 25

This bit is set when the cell is partially built in the current buffer and the BFD_BuffCont bit in the Buffer Descriptor (page 5-19) is set indicating the buffer is continued but there is no buffer available.

BFS_ErrCPI **CPI Error (BFS_ConnNum Cleared) 25**

This bit is set when the CPI field of the AAL5 PDU is not zero.

BFS_ErrLowMem

One Free Buffer List Empty 24

This bit is set when the processor attempts to retrieve a buffer from a free list and either the small or large list is empty.

BFS_ErrFreeSel

Free Select Error (BFS_ConnNum Set) 23

This bit is set when a buff command with the BFS_BuffFree bit set and BFS_FreeSel field equal to 6 or 7 is issued, or when an RxCell command that requires a free buffer and a VCD_FreeSel field of 6 or 7 is issued.

BFS_ErrAbort

Zero Payload (BFS_ConnNum Cleared) 23

This bit is copied from the Buffer Descriptor and is set in the receive direction when the payload field of an EOM cell holds a zero value.

BFS_ZeroConNum

Zero Connection Error (BFS_ConnNum Set) 22

This bit is set when the TxConClose, RxConClose, or RxCell command is issued with a zero connection number.

BFS_ErrLength

Payload Length Error (BFS_ConnNum Cleared) 22

This bit is copied from the Buffer Descriptor. The EDMA sets this bit if an incorrect payload length is detected. The EDMA detects lost or misinserted cells by comparing the accumulated payload length, LenVCD, with the payload length, LenCell, extracted from the CS-PDU trailer. The accumulated payload length is the number of received cells since the last EOM cell multiplied by 48 (including the current EOM cell.) The BFD_ErrLength bit is set if the following relation is not detected (receive direction only):

$$LenCell + 8 \leq LenVCD < LenCell + 56$$

BFS_ErrNotOpen

VCD Closed (BFS_ConnNum Set) 21

This bit is set when the EDMA receives a TxCell or RxCell command for a VC which has the VCD_ConOpen bit cleared.

BFS_ErrCrc **CRC32 Error (BFS_ConnNum Cleared) 21**

This bit is copied from the Buffer Descriptor. The EDMA sets this bit in the receive direction when the computed CRC32 does not match the one extracted from an EOM cell.

BFS_FreeSel[2:0]

Free List Select [20:18]

These bits are copied from the BFD_FreeSel field. They represent the free list to which the buffer belongs.

BFS_ConClose

Connection Closed (BFS_ConnNum Set) 17

This bit indicates that the Tx or Rx connection for which the TxConClose/RxConClose command was issued is closed.

BFS_BuffFree

Buffer Origin (BFS_ConnNum Cleared) 17

This bit indicates the buffer origin. When the bit is cleared, the buffer was attached by the user to a VC Descriptor. When the bit is set, the buffer was taken from a Free Buffer List and the BFS_BuffLarge bit specifies whether it was from the Free Large or Small Buffer List. The BFS_BuffFree bit can only be set in the receive direction.

BFS_Cell Hold

Completion Message Due to Cell Hold (BFS_ConnNum Set) 16

When this bit is set, the completion message in the Tx/Rx Completion Queue is due to the Cell Hold mode of the VC Descriptor.

BFS_BuffLarge

Buffer from Large or Small Free Buffer List (BFS_ConnNum Cleared) 16

When this bit is set, the buffer was taken from the Large Free Buffer List (if BFS_BuffFree is also set). When this bit is cleared, the buffer was taken from the Small Free Buffer List (if BFS_BuffFree is set).

The completion messages returned in the Transmit, Receive, and Buffer Completion Queues are summarized in Table 5.7, Table 5.8, and Table 5.9.

Table 5.7 Tx Completion Queue Messages

Message			
Primary	**Status Bits**	**Auxiliary**	**Description**
buff	0x0000 (BFD_BuffCont Clear) 0x2000 (BFD_BuffCont Set)	conn	Buffer Completion
conn	0x4001	cell[1]	Cell Hold
conn	0xC800	cell[2]	ErrNoData
conn	0xC200	cell[2]	ErrNoContBuff
conn	0x4002	0	TxConclose
0 (conn)	0xC042	0	TxConclose with zero connection error
conn	0xC020	cell[2]	ErrNotOpen
buff	0x8400	conn	Payload Overrun

1. The cell is not sent to the ACI TxFIFO.
2. The cell is returned to the ACI freelist.

Table 5.8 Rx Completion Queue Messages

Message			
Primary	**Status Bits**	**Auxiliary**	**Description**
buff	0x0002 (BFD_BuffCont Clear, Free Small) 0x0003 (BFD_BuffCont Clear, Free Large) 0x2002 (BFD_BuffCont Set, Free Small) 0x2003 (BFD_BuffCont Set, Free Large) 0x8080 (ErrAbort in PDU) 0x8040 (ErrLength in PDU) 0x8020 (ErrCrc in PDU) 0x8100 (ErrLowMem)	conn	Buffer Completion
conn	0x4001	cell[1]	Cell Hold
conn	0xC400	cell[2]	ErrNoMem
conn	0xC098 (Free Sel = 6) 0xC09C (Free Sel = 7)	cell[2]	Err Free Sel
conn	0xD000	buff	Err zero pointers in BFD
conn	0x4002	0	RxConClose

Table 5.8 Rx Completion Queue Messages (Cont.)

Message			
Primary	**Status Bits**	**Auxiliary**	**Description**
0 (conn)	0xC040	cell[2]	RxCell with zero connection error
0 (conn)	0xC042	0	RxConclose with zero connection error
conn	0xC020	cell[2]	ErrNotOpen

1. The cell is not returned to the ACI freelist.
2. The cell is not returned to the ACI freelist if the error is reported.

Table 5.9 Buff Completion Queue Messages

Message			
Primary	**Status Bits**	**Auxiliary**	**Description**
conn	0x4000	buff	Connection Reactivation
0 (conn)	0xC098 (Free Sel = 6) 0xC09C (Free Sel = 7)	buff	Err Free Select
conn	0xD000	buff	Err zero pointers in BFD
conn	0xE000	buff	Err zero BuffSize in BFD
buff	0x8800	conn	ErrNotOpen
0 (conn)	0xC040	buff	Err ZeroConNum

5.4 Data Structure Locations

The EDMA's three primary data structures (VC Descriptors, Buffer Descriptors, and buffers) can be located in three different address spaces; Primary Port memory, Secondary Port memory, or Cell Buffer Memory. This section describes the system options for locating these data structures.

5.4.1 VC Descriptors Address Calculation

VC Descriptors are referenced using 16-bit wide Connection Numbers. The VC Descriptors are located in the Primary Port memory space, Secondary Port memory space, or in Cell Buffer Memory. In addition, to further reduce the Secondary Port bandwidth requirements, you can place a limited number of VC Descriptors in Cell Buffer Memory. These are the initial VCDs as defined by the SCD_VCDinCB field of the SCD_Ctrl register (see Section 7.6.8, “Calculating a VC Descriptor Address.”)

To compute the VC Descriptor address, the EDMA adds the contents of the VCD_Base field of the Tx_EDMA_VCD_Base register or Rx_EDMA_VCD_Base register and the Connection Number field of the EDMA_TxConNum or EDMA_RxConNum register. Figure 5.21 shows the format of the Tx/Rx_EDMA_VCD_Base register. They are located at address 0xB800.00CC and 0xB800.00D8 respectively. For backward compatibility, the Rx_EDMA_VCD_Base register is dual addressed with the Tx_EDMA_VCD_Base register. Both EDMA_VCD_Base registers are written with the same data when the APU writes to the Tx_EDMA_VCD_Base register. When the APU writes to 0xB800.00D8, only the Rx_EDMA_VCD_Base register is updated. Due to layout restrictions, this register is not readable.

Figure 5.21 TX/RX_EDMA_VCD_Base Register

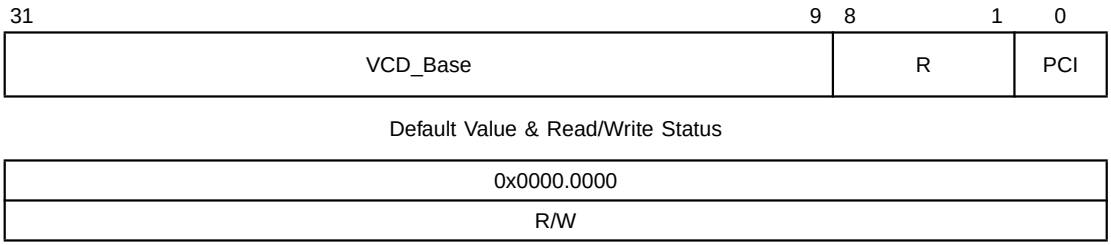


Figure 5.22 shows the calculation for a VCD in PCI memory. When VCDs are located in PCI memory, bit 0 of the Tx/Rx_EDMA_VCD_Base register should be set. Then bits [31:9] of the Tx/Rx_EDMA_VCD_Base register are extended with trailing zeros and added to bits [20:5] of the EDMA_Tx/RxConNum register extended with trailing zeros.

Figure 5.22 VC Descriptor Address Calculation for PCI Memory

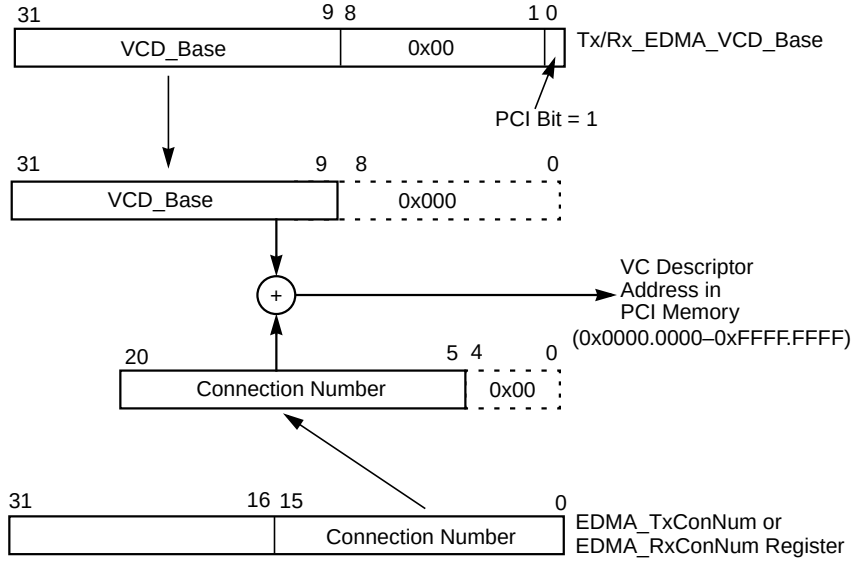
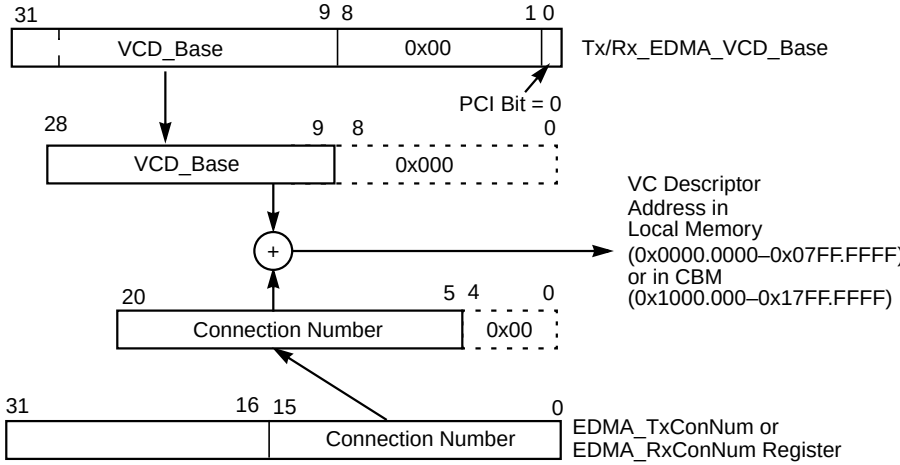


Figure 5.23 shows the address calculation for VCDs in Local or Cell Buffer Memory. When VCDs are in Local Memory or CBM, bit 0 of the Tx/Rx_EDMA_VCD_Base register is cleared. Then bits [28:9] of the Tx/Rx_EDMA_VCD_Base register are extended with trailing zeros and added to bits [20:5] of the EDMA_Tx/RxConNum register extended with trailing zeros.

Figure 5.23 VC Descriptor Address Calculation for Local or Cell Buffer Memory



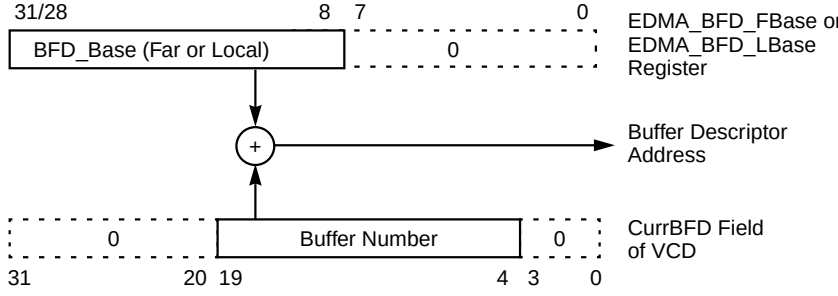
Note: Bit 28 of VCD_Base determines whether address is in Local Memory or CBM. Bits [31:29] of VCD_BASE are don't cares.

Note: The results of the VCD address calculation in the ranges 0x0800.0000–0x0FFF.FFFF and 0x1800.0000–0x1FFF.FFFF, when the PCI bit = 0, cause bus error interrupts.

5.4.2 Buffer Descriptors

Buffer Descriptors can be located in either the Primary Port memory or the Secondary Port memory, and they are referenced using a 16-bit wide Buffer Number. Figure 5.24 illustrates the Buffer Descriptor address computation scheme.

Figure 5.24 Buffer Descriptor Address Calculation



If Buffer Descriptors are located in PCI memory, bits [31:8] of the EDMA_BFD_FBase register are added to the Buffer Number to access them. If BFDs are located in Secondary Memory, bits [28:8] of the EDMA_BFD_LBase register are added to the Buffer Number to access them. Figure 5.25 shows the format of the EDMA_BFD_FBase register (address 0xB800.00D4) and Figure 5.26 shows the format of the EDMA_BFD_LBase register (address 0xB800.00D0).

Figure 5.25 EDMA_BFD_FBase Register

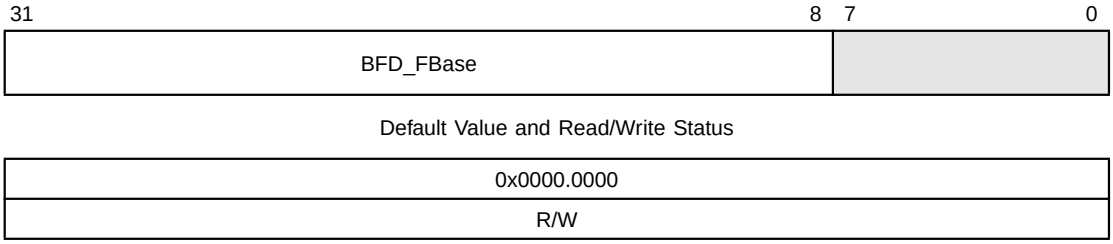
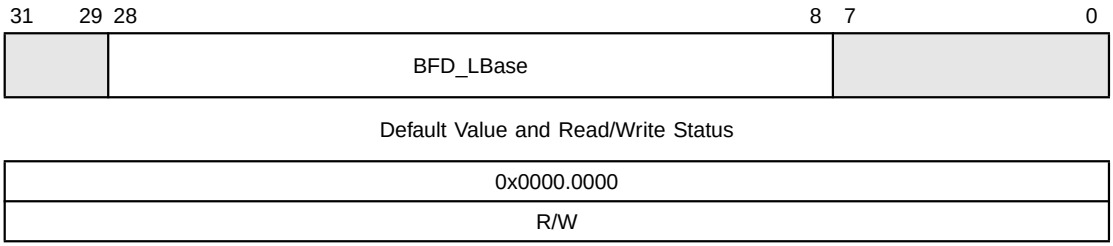


Figure 5.26 EDMA_BFD_LBase Register



Although the number of times the EDMA must access the Buffer Descriptors has been minimized, processing linked lists of Buffer Descriptors can require more than one access. For applications where Buffer Descriptors are placed in shared memory with high latency, it is possible to reduce shared Buffer Descriptor accesses even further by correctly setting the EDMA_TxBFD_Copy and/or EDMA_RxBFD_Copy bits in the EDMA_Ctrl register (page 5-52).

When a EDMA_Tx/RxBFD_Copy bit is cleared, the EDMA always uses the same base register for the Buffer Descriptors. The EDMA_Tx/RxBFD_Far bit in the EDMA_Ctrl register (page 5-52) specifies if Far Base (when set) or Local Base (when cleared) is used.

When the EDMA_Tx/RxBFD_Copy bit is set, the EDMA_Tx/RxBFD_Far bit is not used, and the EDMA processors make Buffer Descriptor copies when necessary.

For the transmit direction, the Buff Processor copies Buffer Descriptors from the PCI address to the Secondary address when a buff command is executed. For the receive direction, when a buffer is completed, the RxCell Processor copies Buffer Descriptors from the Secondary address to the PCI address before placing the Buffer Number in the Receive Completion Queue. All other accesses to the Buffer Descriptor use a local address.

This structure makes it possible to address three different applications separately for the transmit and receive sides. When the external host uses different Buffer Descriptor formats than the EDMA, the APU has to perform the necessary translation. The APU places the Buffer Descriptor in the Secondary Port memory and clears both the EDMA_Tx/RxBFD_Copy and EDMA_Tx/RxBFD_Far bits. If the formats are compatible and the shared bus of the Primary Port is fast, the Buffer Descriptor copy is not required. The APU then clears the EDMA_Tx/RxBFD_Copy bit and sets the EDMA_Tx/RxBFD_Far bit. If the shared bus is slow, the APU can set the EDMA_Tx/RxBFD_Copy bit.

5.4.3 Buffer Payload

The buffer payload can be located in either Primary Port memory or Secondary Port memory. It is referenced using a 32-bit or a 27-bit wide address pointer, pBuffData.PCI or pBuffData.Sec, placed in the Buffer Descriptors. In the VC Descriptors, pBuffData is a 33-bit pointer. If pBuffData[32] is set, the buffer payload is in PCI memory.

In applications where shared memory located on the Primary Port has low latency, the EDMA processors may access the buffers placed in that shared memory on a per-cell basis. In that case, the EDMA processors access the shared memory to transfer the required cell payload of up to 48-bytes for each TxCell or RxCell command. If shared memory has a high latency, it is possible to automatically copy the whole buffer between shared memory and local memory (attached to the Secondary Port).

For the transmit direction, if the pBuffData.PCI and PBuffData.Sec fields in the BFD are nonzero, the EDMA performs the buffer copy operation when a Buffer Descriptor is attached to a VC Descriptor. The

EDMA retrieves the source address from the Buffer Descriptor's pBuffData.PCI field and the destination address from the pBuffData.Sec field. The number of bytes transferred is retrieved from the Buffer Descriptor's BuffSize field.

For the receive direction, if the pBuffData.PCI and PBuffData.Sec fields are nonzero, the EDMA performs the buffer copy operation when a buffer is completed and before the Buffer Number is placed in the Receive Completion queue. The EDMA retrieves the source address from the Buffer Descriptor's pBuffData.Sec field and the destination address from the pBuffData.PCI field. The number of bytes transferred is retrieved from the Buffer Descriptor's BuffSize field.

The Buff and RxCell Processors generate internal commands to the Move Processor. The Buff and RxCell Processors are stalled until the Move Processor completes the transfer.

5.5 Register Descriptions

The registers listed in Table 5.10 provide the mechanism for controlling EDMA operations. The APU accesses these registers at memory address 0xB800.00XX, where XX is specified in the Offset column.

Table 5.10 EDMA Memory Mapped Registers

Name	Offset	Size	R/W	Description
EDMA_TxCompl	0x00	32	R	Read Transmit Completion Queue
EDMA_TxConNum	0x04	32	R/W	Connection Number for the TxCell Command
EDMA_TxCell	0x08	32	R/W	Issue a TxCell Command
EDMA_TxConAct	0x10	32	R	Current active ConNum processed by TxCell Processor
EDMA_TxComplB	0x14	32	R	Auxiliary Transmit Completion Queue
EDMA_TxConClose	0x18	32	W	Transmit Connection Close Command
EDMA_AAL5Pad	0x1F	8	R/W	AAL5 Pad Byte
EDMA_RxCompl	0x40	32	R	Read Receive Completion Queue
(Sheet 1 of 3)				

Table 5.10 EDMA Memory Mapped Registers (Cont.)

Name	Offset	Size	R/W	Description
EDMA_RxConNum	0x44	32	R/W	Connection Number for the RxCell Command
EDMA_RxCell	0x48	32	R/W	Issue an RxCell Command
EDMA_RxConAct	0x50	32	R	Current Active ConNum Processed by RxCell Processor
EDMA_RxComplB	0x54	32	R	Auxiliary Receive Completion Queue
EDMA_RxConClose	0x58	32	W	Receive Connection Close Command
EDMA_Buff	0x80	32	R/W	Issue a buff Command
EDMA_BuffCompl	0x88	32	R	Read Buff Completion Queue
EDMA_BuffComplB	0x8C	32	R	Auxiliary Buff Completion Queue
EDMA_BuffConAct	0x90	32	R	Current Active ConNum Processed by Buff Processor
EDMA_LBuff0	0x94	16	R/W	Head of Large Free Buffer List 0
EDMA_SBuff0	0x96	16	R/W	Head of Small Free Buffer List 0
EDMA_LBuff1	0x98	16	R/W	Head of Large Free Buffer List 1
EDMA_SBuff1	0x9A	16	R/W	Head of Small Free Buffer List 1
EDMA_MoveSrc	0xA0	32	R/W	Program the Source Address for a move Command
EDMA_MoveDst	0xA4	32	R/W	Program the Destination Address for a move Command
EDMA_MoveCount	0xA8	32	R/W	Issue a move Command
EDMA_MoveCount2	0xAC	32	R/W	Issue Enhanced move Command
EDMA_LBuff2	0xB0	16	R/W	Head of Large Free Buffer List 2
EDMA_SBuff2	0xB2	16	R/W	Head of Small Free Buffer List 2
EDMA_LBuff3	0xB4	16	R/W	Head of Large Free Buffer List 3
EDMA_SBuff3	0xB6	16	R/W	Head of Small Free Buffer List 3
EDMA_LBuff4	0xB8	16	R/W	Head of Large Free Buffer List 4
EDMA_SBuff4	0xBA	16	R/W	Head of Small Free Buffer List 4
EDMA_LBuff5	0xBC	16	R/W	Head of Large Free Buffer List 5

(Sheet 2 of 3)

Table 5.10 EDMA Memory Mapped Registers (Cont.)

Name	Offset	Size	R/W	Description
EDMA_SBuff5	0xBE	16	R/W	Head of Small Free Buffer List 5
EDMA_Ctrl	0xC0	16	R/W	EDMA Control Bits
EDMA_Status	0xC4	16	R	See the EDMA_Status Register Description on page 5-33
EDMA_LBuffSize	0xC8	16	R/W	Size of Large Buffers in Bytes
EDMA_SBuffSize	0xCA	16	R/W	Size of Small Buffers in Bytes
EDMA_VCD_Base	0xCC	32	R/W	Base Address of the VC Descriptor Table
EDMA_BFD_LBase	0xD0	32	R/W	Local Base Address of the Buffer Descriptor Table
EDMA_BFD_FBase	0xD4	32	R/W	Far Base Address of the Buffer Descriptor Table
EDMA_ErrMask	0xDC	16	R/W	Error Mask Register
EDMA_BusErr	0xE3	8	R	Address and Bus Error Register
(Sheet 3 of 3)				

The EDMA_TxConAct, EDMA_RxConAct and EDMA_BuffConAct registers contain either the current active Connection Number (that the TxCell, RxCell, or Buff Processor will process) or, if the module is idle, the value zero.

The EDMA-AAL5Pad register holds the pad byte for the AAL5 CPCS-PDU. It is cleared at reset.

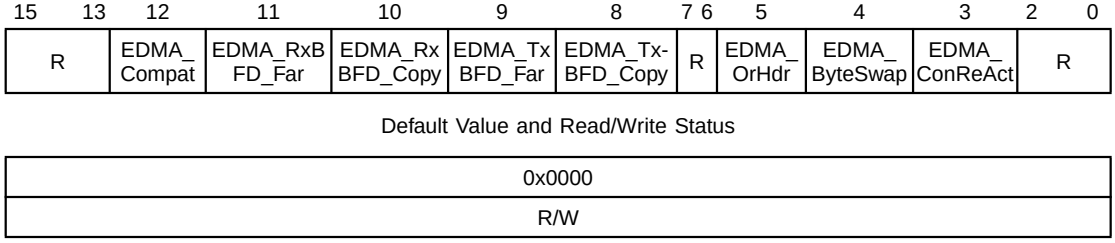
The EDMA_LBuffSize and EDMA_SBuffSize registers specify the sizes of large/small buffers to be used when a buffer is linked from a Free Buffer List. The values in both the EDMA_LBuffSize and EDMA_SBuffSize register must be larger than or equal to 48 for correct EDMA operations. The values present in the `BuffSize` field of the Buffer Descriptor are ignored at that time as they were most probably overwritten by the actual number of bytes received the last time the buffer was used. Instead, the EDMA uses the value in the EDMA_LBuffSize or EDMA_SBuffSize register, sets the `BFS_BuffFree` bit in the EDMA_BuffCompl register (page 5-26) and sets or clears the `BFS_BuffLarge` bit in the EDMA_BuffCompl register to indicate the origin of the buffer.

The EDMA_LBuff0–5 and EDMA_SBuff0–5 registers point to the beginning of Large and Small Free Buffer Lists. They are accessible to software mainly for test and diagnostic purposes.

5.5.1 EDMA Control Register

The EDMA_Ctrl register contains the control bits shown in Figure 5.27. The register is located at address 0xB800.00C0.

Figure 5.27 EDMA_Ctrl Register



R **Reserved** **[15:13]**
Not used in the L64364.

EDMA_Compat
Compatibility **12**
When cleared, this bit configures the move engine to operate in L64363 compatibility mode (move engine uses big endian addressing for EDMA initiated moves). When this bit is set and the EDMA_ByteSwap bit is set, the move engine uses little endian addressing for EDMA initiated moves.

EDMA_Rx/TxBFD_Far and EDMA_Rx/TxBFD_Copy **[11:8]**
These bits specify the location of Buffer Descriptors and whether the Buffer Descriptors are copied to/from the far address as described in Section 5.4.2, "Buffer Descriptors."

R **Reserved** **[7:6]**
Not used in the L64364.

EDMA_OrHdr **OR Header** **5**
This bit controls the generation and extraction of the EFCI and CLP cell header bits. If EDMA_OrHdr is set, these bits

are logically ORed over a CS-PDU (see the description of the BFD_EFCI and BFD_CLP bits in Section 5.2.2.2, “Buffer Descriptor Control Field”). If the bit is cleared, the VCD_CLP and VCD_EFCI represent the bits extracted from the last received cell or the bits inserted into all transmitted cells.

EDMA_ByteSwap

Byte Swapping **4**

This bit controls byte swapping for cell payload transfers. Byte swapping assures proper interfacing between the ATMizer II+ chip and a little endian external bus master. When the EDMA_ByteSwap bit is set, the EDMA swaps bytes in the cell payload during the execution of a TxCell or RxCell command. The placement of bytes depends on the EDMA_ByteSwap bit and the value of pBuffData as described in Section 5.6.4, “Big Endian and Little Endian.” Byte swapping is never done for control structures (VC Descriptors and Buffer Descriptors).

EDMA_ConReAct

Reactivate Connection **3**

This bit enables connection reactivation in the transmit direction. In many applications, you may want to remove the connection from scheduling tables when there is no data to send and then reactivate the connection when data becomes available. To support this operation, the EDMA Buff Processor can flag when a connection that had no data to send receives buffer data for segmentation.

When the EDMA_ConReAct bit is set and VCD_BuffPres bit in the VC Descriptor (page 5-12) changes from 0 to 1 as a result of the buff command, the EDMA places the Connection Number in the Buff Completion Queue so that the APU may reschedule the connection for service.

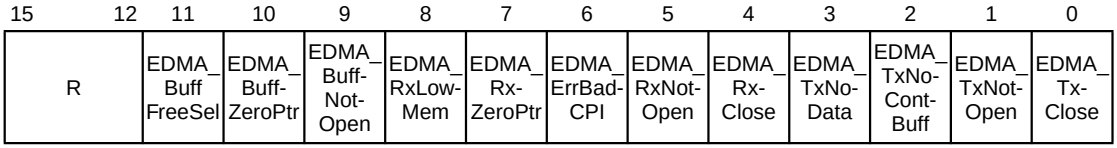
R **Reserved** **[2:0]**

Not used in the L64364.

5.5.2 EDMA Error Mask Register

The EDMA_ErrMask register is used to mask the errors that are reported in the completion queues. Clearing the bits masks the errors. The register is at memory address 0xB800.00DC. The format of the register is shown in Figure 5.28.

Figure 5.28 EDMA_ErrMask Register



15 12	11	10	9	8	7	6	5	4	3	2	1	0
R	EDMA_ Buff FreeSel	EDMA_ Buff-ZeroPtr	EDMA_ Buff-Not-Open	EDMA_ RxLow-Mem	EDMA_ Rx-ZeroPtr	EDMA_ ErrBad-CPI	EDMA_ RxNot-Open	EDMA_ Rx-Close	EDMA_ TxNo-Data	EDMA_ TxNo-Cont-Buff	EDMA_ TxNot-Open	EDMA_ Tx-Close

Default Values & Read/Write Status

	0xFFF
	R/W

R **Reserved** **[15:12]**
Not used in the L64364.

EDMA_BuffFreeSel
FreeSel Field Equal to Command **11**
This bit is set to enable the BFS_ErrFreeSel field in the Buff completion queue. BFS_ErrFreeSel bit is set when the buff command is issued with the BFS_BuffFree bit set and the BFS_FreeSel field equal to 6 or 7.

EDMA_BuffZeroPtr
Zero Pointer in BFD on Buff Command **10**
This bit is set to enable the BFS_ErrZeroPtr completion message when a buff command is issued. BFS_ErrZeroPtr occurs when both the pBuffData.Sec and pBuffData.PCI pointers in the BFD are zero.

EDMA_BuffNotOpen
Buff Command to a Closed Connection **9**
This bit is set to enable the BFS_ErrNotOpen completion message when a buff command is issued to a VC Descriptor with the VCD_ConClose bit cleared.

EDMA_RxLowMem
Low Memory Error **8**
This bit is set to enable the BFS_ErrLowMem message in the Rx completion queue when one of the free lists is empty.

EDMA_RxZeroPtr
Zero Pointer in BFD on RxCell Command **7**
This bit is set to enable the BFS_ErrZeroPtr completion message when an RxCell command is issued.

BFS_ErrZeroPtr is set when both the pBuffData.Sec and pBuffData.PCI pointers in the BFD are zero.

EDMA_ErrBadCPI

ErrBadCPI in Buff Format **6**

This bit is set to enable the BFS_ErrBadCPI error reporting in Buff format.

EDMA_RxNotOpen

RxCell Command to a Closed Connection **5**

This bit is set to enable the BFS_ErrNotOpen completion message when an RxCell command is issued to a VC Descriptor with the VCD_ConClose bit cleared.

EDMA_RxClose

Completion of RxConClose Command **4**

This bit is set to enable the completion message for the RxConClose command to a receive VC Descriptor.

EDMA_TxNoData

ErrNoData on TxCell Command **3**

This bit is set to enable the BFS_ErrNoData completion message for the TxCell command to a transmit VC Descriptor.

EDMA_TxContBuff

ErrNoContBuff on TxCell Command **2**

This bit is set to enable the BFS_ErrNoContBuff completion message for the TxCell command to a transmit VC Descriptor.

EDMA_TxNotOpen

TxCell Command to a Closed Connection **1**

This bit is set to enable the BFS_ErrNotOpen completion message when a TxCell command is issued to a VC Descriptor with the VCD_ConClose bit cleared.

EDMA_TxClose

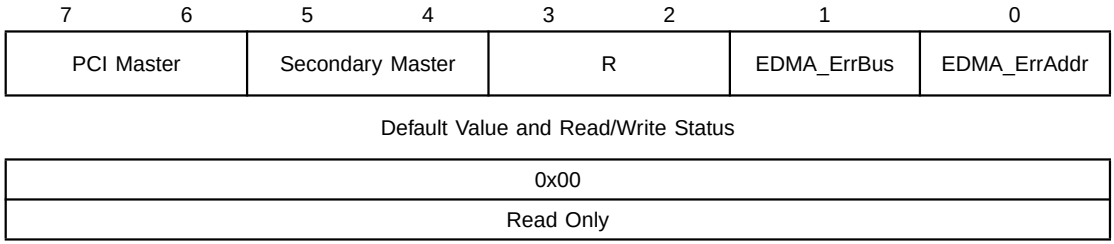
Completion of TxConClose Command **0**

This bit is set to enable the completion message for the TxConClose command to a transmit VC Descriptor.

5.5.3 EDMA Bus Error Register

The EDMA_BusErr register is used to determine the cause of the `IntEDMA_BusErr` interrupt (see Section 4.8.1, “External Nonvectored Interrupts”) and has the fields shown in Figure 5.29. The register is at address 0xB800.00E3.

Figure 5.29 EDMA_BusErr Register



PCI Master [1:0]

Current PCI Bus Master **[7:6]**

These bits indicate which ATMizer II+ engine is the current PCI Bus master as follows:

PCI Master [1:0]	Definition
0b00	Port idle
0b01	Tx Engine
0b10	Rx Engine
0b11	Buff Engine

Secondary Master [1:0]

Current Secondary Bus Master **[5:4]**

These bits indicate which ATMizer II+ engine is the current Secondary Bus master as follows:

Sec. Master [1:0]	Definition
0b00	Port idle
0b01	Tx Engine
0b10	Rx Engine
0b11	Buff Engine

R **Reserved** **[3:2]**
Not used in the L64364.

EDMA_ErrBus
Bus Error **1**
This bit is set when the EDMA has a bus error.

EDMA_ErrAddr
Address Error **0**
This bit is set when the EDMA has an address error.

5.6 AAL5 Mode Operation

The EDMA is a custom Segmentation and Reassembly multiprocessor that executes a complex microcoded program. This section explains the operation of the EDMA using a stylized pseudo-code.

To simplify the description, the pseudo-code does not handle the case when the CS-PDU payload is fragmented between multiple buffers. The hardware EDMA module handles that case transparently.

5.6.1 Transmit Cell Processing Requests

The EDMA TxCell Processor retrieves a VC Descriptor Number (ConNum) and a Cell Number (CellNum) from the EDMA TxCell Request Queue.

```
1. read VC Descriptor
2. if BuffDone
3.        insert cell header with EOM
4.        clear cell bytes 4 to 45
5.        place CS-PDU payload length in bytes 46-47
6.        place final crc32 in bytes 48-51
7.        send cell out
8.        clear BuffDone bit
9.        delink current buffer
10.       return current buffer to Completion Queue
11. if next buffer present
12.             install next buffer
13.       exit
14. if no buffer installed
15.       exit
```

```
16. N = min(48, Nbytes)
17. transfer N bytes from buffer to cell and update crc32
18. if current buffer exhausted
19.       if N > 40
20.             insert cell header, no EOM
21.             fill rest of the cell with AAL5Pad byte
22.             send cell out
23.             set BuffDone bit
24.       else
25.             insert cell header with EOM
26.             fill cell bytes with AAL5Pad till byte 45
27.             place CS-PDU payload length in bytes 46-47
28.             place final crc32 in bytes 48-51
29.             send cell out
30.             delink current buffer
31.             return current buffer to Completion Queue
32.             if next buffer present
33.             install next buffer
34.       else
35.       insert cell header, no EOM
36.       send cell out
37. update VC Descriptor
```

Lines 2–13 – handle the case where the entire CS-PDU payload was sent, except a cell with padding and a CS-PDU trailer still needs to be sent.

Lines 16–17 – transfer the payload from a buffer to the Cell Buffer Memory.

Lines 20–23 – handle the case where the CS-PDU trailer would not fit in the current cell.

Lines 25–33 – process the case where the trailer fits and an EOM cell should be sent.

Delinking the current buffer operation (line 30) consists of advancing the `CurrBFD` index to point to the following Buffer Descriptor. Installing the next buffer (line 33) consists of copying the `BuffSize` and `pBuffData` fields from the Buffer Descriptor to the VC Descriptor and performing various initialization routines.

Note: The EDMA does not modify Buffer Descriptors for the transmit direction. In particular the NextBFD field in the Buffer Descriptor returned to the EDMA Completion Queue contains a possibly invalid pointer to the next Buffer Descriptor.

5.6.2 Receive Cell Processing Requests

The EDMA RxCell Processor retrieves the VC Descriptor Number (ConNum) and a Cell Number (CellNum) from the EDMA RxCell Request Queue.

```
1. read VC Descriptor
2. if no buffer installed
3. if buffer available
4.        install buffer
5.        else
6.        pull buffer from small free buffer list
7.        install buffer
8. read cell header from Cell Buffer Memory
9. set N = 48
10.if an EOM cell
11.       read expected payload length from the cell
12.if expected payload length different from accumulated
13.       return buffer with error status
14.       compute N (number of bytes in cell payload)
15.       read expected crc32
16.       if Number-Of-Bytes-In-Buffer + N > Buffer Size
17.       delink current buffer
18.       return the current buffer in completion queue
19.       if next buffer available
20.       install next buffer
21.             else
22.       pull buffer from large free buffer list
23.       install buffer
24.       transfer N payload bytes to buffer
25.       if EOM cell
26.       update crc32 with padding and length
27.       check crc32 against expected and set error codes
28.       delink current buffer
29.       return current buffer
```

Lines 2–8 – process the case where the cell is the first cell of the payload and the receiving buffer is not yet available. Line 8 checks whether the current cell is a continuation cell or an EOM cell.

Line 9 – the expected number of bytes to transfer is preset to 48 for a continuation cell. For an EOM cell, the EMDA retrieves the payload length and the expected CRC32 from the cell and then computes the number of bytes to transfer based on the payload length. The expected payload length is compared with the accumulated length to detect lost cells.

Lines 16–23 – check whether there is enough space in the current buffer to transfer the cell payload and, if necessary, returns the current buffer to the completion queue and opens a new buffer.

Line 24 – the EDMA transfers the cell payload and performs CRC32 checks for EOM cells.

The EDMA modifies Buffer Descriptors for the receive direction. It inserts all status codes in the `BFD_Ctrl` field, places the Connection Number in the `ConNum` field, and writes the number of bytes received in the `BuffSize` field.

5.6.3 Free Buffers

The EDMA maintains twelve lists of buffers, six pairs of Small Free Buffer Lists and Large Free Buffer Lists. The buffers from a free list are used in the receive direction if the CS-PDU needs to be reassembled and there is either no space in the current buffer or no buffer is attached.

A buffer from the Small Free Buffer List is used at the beginning of the CS-PDU. If additional buffers are required they are taken from the Large Free Buffer List. The Rx VCD has a field that indicates which free list should be accessed when a buffer is required. The Free Buffer Lists are initialized by the EDMA_Lbuff0–5 and EDMA_Sbuff0–5 registers.

Buffers are returned to a free list whenever the `BFS_BuffFree` bit in the EDMA_Buff register (page 5-26) is set. The EDMA examines the `BFS_BuffLarge` and the `BFS_FreeSel` bits in the EDMA_Buff register to decide which buffer list it should use. A free buffer is always inserted at the beginning of the list and is the first one reused. This arrangement is different from the instance when a buffer is attached to a VC Descriptor

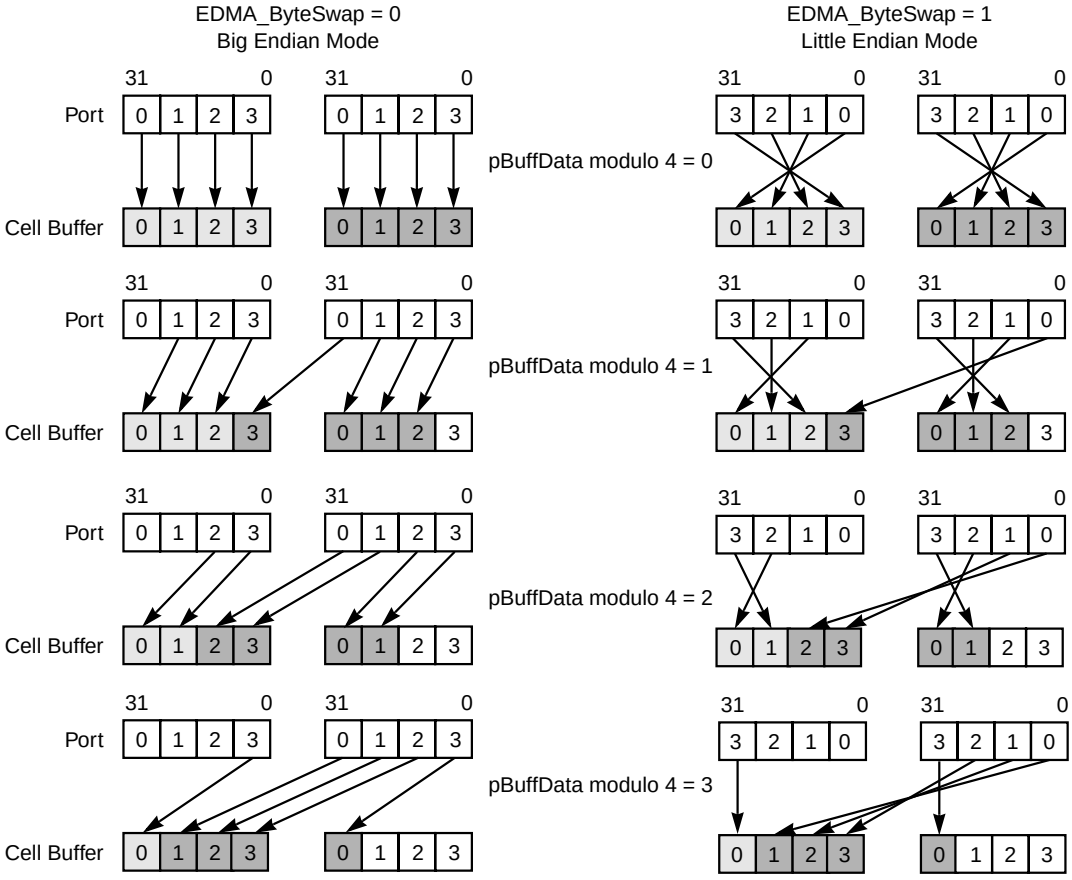
where it is appended to the tail of the list. However, appending to the end of a list requires two pointers, a head and a tail pointer. For free buffers, the list order is not important and one pointer can be omitted.

The EDMA places the actual number of received bytes in the BuffSize field of a Buffer Descriptor upon completion. The Buffer Descriptor is then typically passed to a host processor and, when the data is processed, the buffer is returned to a free list. At this point, the APU would have to insert the original buffer size in the BuffSize field. To avoid the extra memory access, the EDMA uses the BFD_BuffLarge bit in the EDMA_Buff register (page 5-26) to determine which free buffer list to use and it takes the buffer size value from an internal register.

5.6.4 Big Endian and Little Endian

The ATMizer II+ chip operates internally in big endian mode. To facilitate interfacing with little endian external bus masters, the EDMA_Ctrl register (page 5-52) contains the EDMA_ByteSwap bit. When this bit is set, the EDMA swaps bytes for the cell payload transfers (TxCell and RxCell commands). Figure 5.30 illustrates the byte swapping.

Figure 5.30 Byte Swapping



5.7 AAL0 Mode Operation

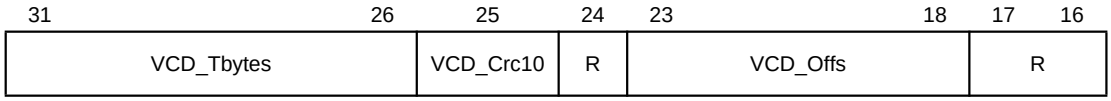
The AAL0 mode can be chosen on a per-VC basis by setting the VCD_AAL0 bit in the VC Descriptor (page 5-11). The AAL0 mode provides hardware support to implement other ATM Adaptation Layers such as AAL1 or AAL3/4 with minimum APU intervention.

The main difference between AAL5 and AAL0 modes involves execution of the Tx/RxCell commands. In the AAL0 mode, the EDMA does not process the AAL5 CS-PDU trailer. The padding, payload length, and CRC32 are not appended to or extracted from a cell.

In the transmit direction, the EDMA copies the cell payload from an external buffer to Cell Buffer Memory and sets or clears the CDS_BOM and CDS_EOM bits in the Cell Descriptor (page 6-5). In the receive direction, the EDMA uses the CDS_EOM bit from the Cell Descriptor to determine if the current cell is an EOM cell. The cell header and Cell Descriptor are copied from the VC Descriptor or from the current buffer depending on the value of VCD_Offs field of the VC Descriptor (Figure 5.31) and on the ACI_CellSize field of the ACI Control register (page 6-10). The EDMA first copies these values from the VC Descriptor. However, if the VCD_Offs is sufficiently small, values from the descriptor for the current buffer overwrite those from the VC Descriptor. In particular, when VCD_Offs is equal to 0, the whole cell including the Cell Descriptor, optional tag bytes, and the cell header are copied from the descriptor for the current buffer.

The unused CRC32 field of the VC Descriptor is used to store additional control information as shown in Figure 5.31 (upper 16 bits).

Figure 5.31 VC Descriptor Control Fields (AAL0 Mode Uses CRC32 Field)



VCD_Tbytes[5:0]
Maximum Transfer Bytes **[31:26]**
This field specifies the maximum number of bytes to transfer (transmit direction only).

VCD_Crc10 **CRC10** **25**
This bit is copied to the CDS_Crc10 field of the Cell Descriptor.

R **Reserved** **24**
Not used in the L64364.

VCD_Offs[5:0] Offset **[23:18]**
This field specifies the offset from the cell location beginning where the data transfer should start.

R **Reserved** **[17:16]**
Not used in the L64364.

In the receive direction, the EDMA uses the value in the CDS_Tbytes field of the Cell Descriptor (page 6-5) to determine how many bytes to store in the buffer. It also uses the CDS_EOM bit from the Cell Descriptor to determine whether or not the current cell is an EOM cell. In the AAL5 mode, bit 0 of the cell header PTI field is used for the EOM check, and the number of bytes to transfer is computed based on the payload length field in the CS-PDU trailer.

In the transmit direction, the EDMA uses the value in the VCD_Tbytes field as the maximum number of bytes to store in Cell Buffer Memory. If the current buffer holds a number of bytes (Nbytes field of the VCD) that is less than or equal to the VCD_Tbytes value and it is the last buffer of a CS-PDU (BFD_BuffCont bit in the Buffer Descriptor is cleared, see page 5-19), the EDMA sets the CDS_EOM bit in the Cell Descriptor. Otherwise, the CDS_EOM bit is cleared. In both cases, the actual number of bytes transferred is stored in the CDS_Tbytes field of the Cell Descriptor. The EDMA clears the CDS_Tbytes field in AAL5 mode.

For the transmit direction, if the VCD_Offs field in the VCD is 0, the Cell Descriptor is copied from the buffer instead of being generated. Similarly, if the VCD_Offs is less than or equal to the position of the cell header (that depends on the number of tag bytes), the cell header is copied from the buffer instead of being copied from the VC Descriptor. This feature allows complete ATM cells in an external buffer (including the cell header) to be transferred to the Utopia Bus. For the receive direction, setting VCD_Offs to a low value allows copying the whole cell (including the Cell Descriptor and the cell header) to an external buffer.

The AAL0 mode can be used to implement the AAL3/4 operation mode. For example, to support AAL3/4 transmit operation, the APU:

- Sets the VCD_AAL0 bit and VCD_CellHold bit in the VC Descriptor control field to 1.
- Sets the VCD_Tbytes field to 44.
- Sets the VCD_Offs field to an offset of 10 (assuming the cell size is 52). The offset of 10 is needed to skip over the Cell Descriptor (4 bytes), the cell header (4 bytes), and the SAR-PDU header (2 bytes).

Chapter 6
ATM Cell Interface

This chapter describes the ATM Cell Interface and includes the following sections:

- Section 6.1, “ACI Overview,” page 6-2
- Section 6.2, “Cell Size and Layout,” page 6-4
- Section 6.3, “Cell Descriptor,” page 6-5
- Section 6.4, “Memory-Mapped ACI Registers,” page 6-9
- Section 6.5, “Cell Buffer Manager,” page 6-18
- Section 6.6, “ACI Receiver,” page 6-21
- Section 6.7, “ACI Transmitter,” page 6-25
- Section 6.8, “Polling Scheme,” page 6-29
- Section 6.9, “Loopback Mode,” page 6-29
- Section 6.10, “Utopia Interface,” page 6-30

Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” should not be modified.

6.1 ACI Overview

The ATM Cell Interface (ACI) handles the transfer of cells between the Cell Buffer Memory (CBM) and the Utopia Port. A block diagram of the ACI is shown in Figure 6.1.

The Utopia Port complies with *The ATM Forum Utopia Level 2, v1.0*, multi-PHY specification. The port operates at 50 MHz with 8-bit data buses and cell-level handshaking.

The ACI includes the following key elements:

- Cell Descriptor
- Memory-Mapped Registers
- Cell Buffer Manager
- ACI Transmitter
- ACI Receiver

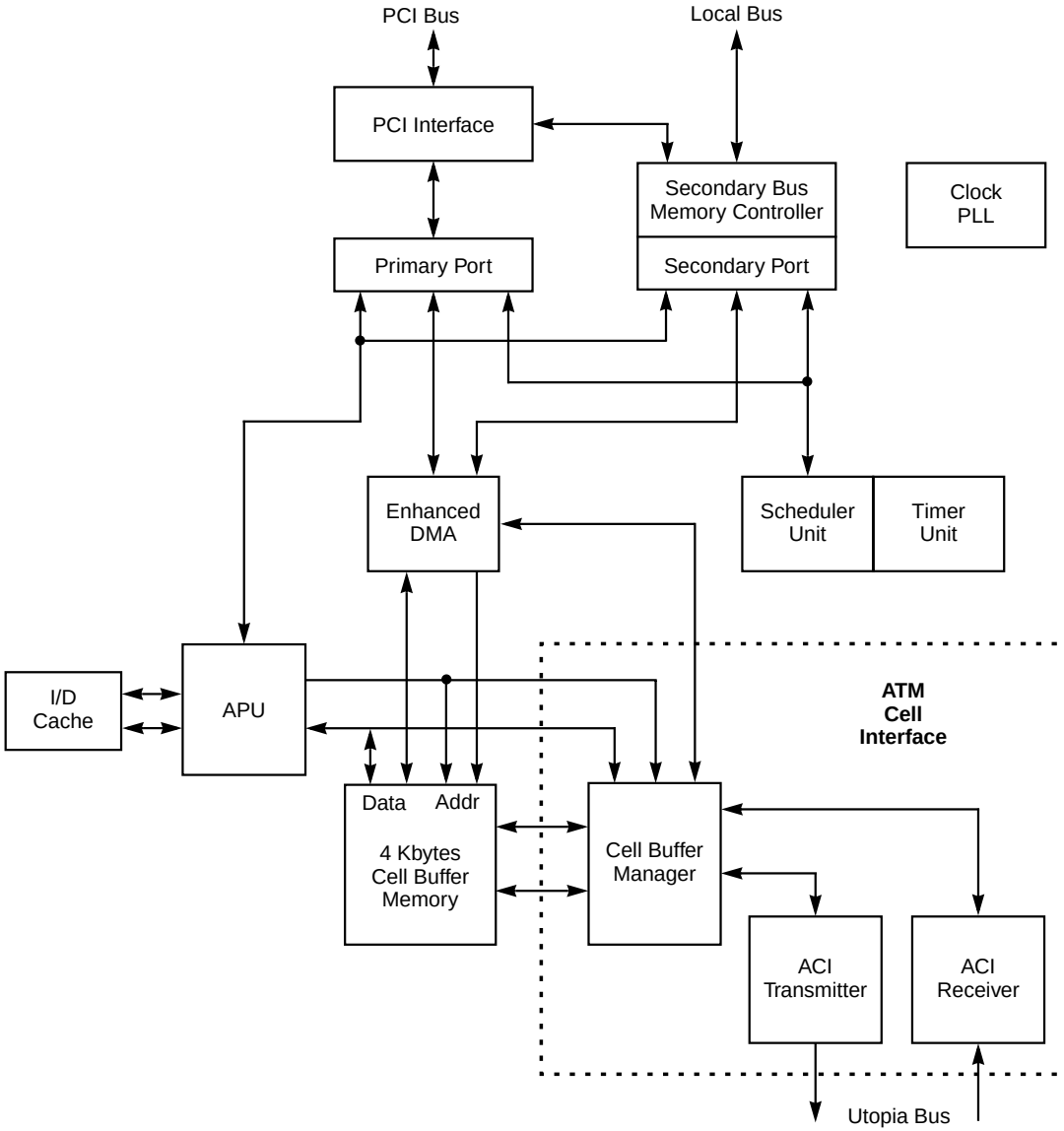
The ACI Transmitter retrieves cells built in Cell Buffer Memory by the EDMA or the APU and sends them, one byte at a time, to a selected Physical Layer (PHY) device. The transmitter can be programmed to generate and insert the Header Error Correction (HEC) byte and to generate and append the CRC10 field.

The ACI Receiver uses a programmed priority scheme to decide which physical device will be serviced in the current cell slot. Then the ACI receives the cell, one byte at a time, and places it in Cell Buffer Memory for further processing by the EDMA or the APU. The receiver can be programmed to verify the HEC byte and CRC10.

Both the transmitter and the receiver may operate as either a Utopia master or slave device.

The ACI uses a 4 byte Cell Descriptor placed in front of a cell in Cell Buffer Memory to manage the transmit and receive operations.

Figure 6.1 ACI Block Diagram



6.2 Cell Size and Layout

Cell size on the Utopia Bus (52-, 56-, 60-, or 64-bytes) is set using the ACI_CellSize field in the ACI_Ctrl register (page 6-10). Since a cell address in CBM must be aligned on a word boundary (4 bytes) and the first 4 bytes are always occupied by a Cell Descriptor, the possible cell sizes in the Cell Buffer Memory are 56-, 60-, 64-, and 68-bytes as shown in Table 6.1.

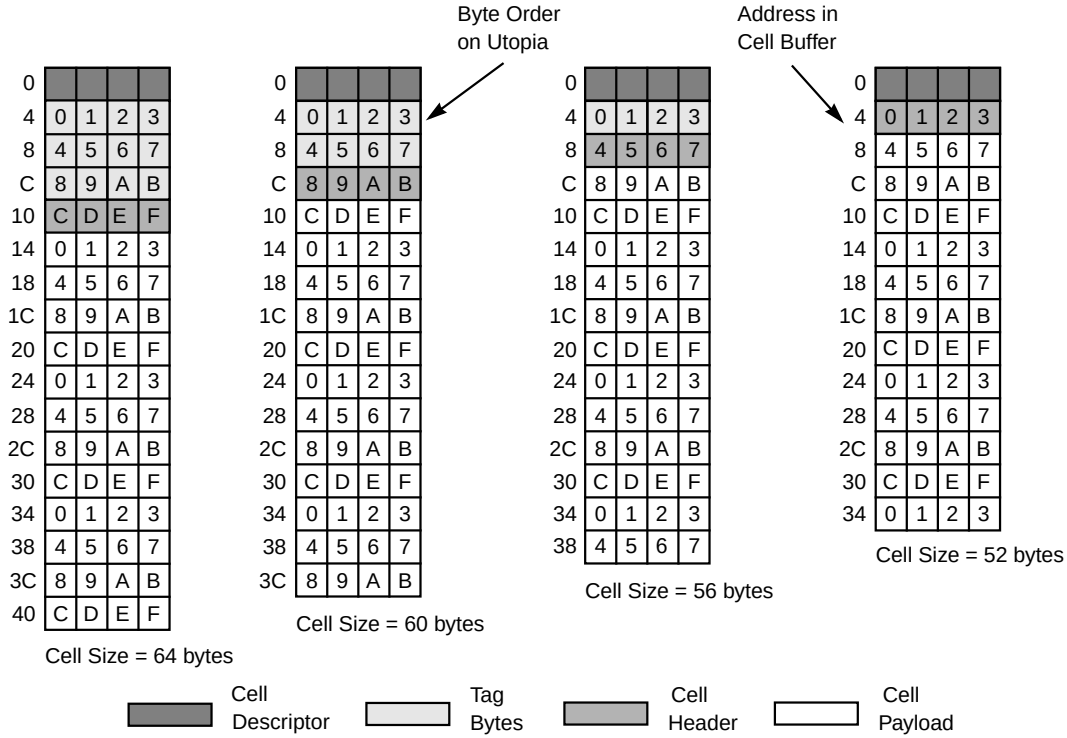
Table 6.1 Cell Size

ACI_CellSize Bits	Cell Size on Utopia (bytes)	Cell Size in CBM (bytes)
0b00	52/53	56
0b01	56/57	60
0b10	60/61	64
0b11	64/65	68

Figure 6.2 illustrates the cell layout. The first 4 bytes of a cell location are used for the Cell Descriptor that is not present on the Utopia Bus. The following 12 bytes are used for the optional tag bytes. Then the next 4 bytes contain the cell header and the last 48 bytes are for the cell payload.

You can program the ACI cell size to support applications with extra header fields. These extra fields use tag bytes to convey additional information.

Figure 6.2 Cell Layout



6.3 Cell Descriptor

The ACI uses a 4 byte Cell Descriptor to manage transmit and receive operations. As shown in Figure 6.2, the Cell Descriptor is located at the front of the cell in CBM. Figure 6.3 shows the Cell Descriptor format; the text that follows Figure 6.3 defines the Cell Descriptor fields.

Figure 6.3 Cell Descriptor Format

31 30	29 18	17 16	15 10	9	8	7	6	5	4 0
R	CDS_Next	R	CDS_TBytes	CDS_ Crc10	CDS_ Par	CDS_ BOM	CDS_ EOM	CDS_ Len	CDS_ PHY

R **Reserved** **[31:30]**
Not used in the L64364.

CDS_Next[11:0]

Next Free Cell **[29:18]**

The Cell Buffer Manager uses this field to build the Receive FIFO, Transmit FIFO, Error FIFO, and the Cell Free List as explained in Section 6.5, “Cell Buffer Manager” page 6-18.

R **Reserved** **[17:16]**

Not used in the L64364.

CDS_TBytes[5:0]

Number of Valid Data Bytes (Transmit) **[15:10]**

The ACI uses this field in the transmit direction to determine how many bytes of cell data are valid. If this field is other than zero, the ACI reads CDS_TBytes bytes of cell payload from Cell Buffer Memory and transfers it to the Utopia Bus. The remaining bytes (48 -CDS_TBytes) of the cell payload are not read from Cell Buffer Memory; instead, they are replaced by a constant value equal to the value of the ACI_ClearBytes register. The ACI_ClearBytes register defaults to a value of zero.

If the CDS_Crc10 bit is set, the last 2 bytes are not cleared. The correct CRC10 value is still inserted in the last 10 bits of the cell (taking into account the cleared bytes of the cell payload) and the preceding 6 bits remain unchanged.

When the CDS_TBytes is cleared, the entire cell payload is transmitted. The EDMA clears the CDS_TBytes field in the AAL5 mode and sets it to the actual number of bytes transferred in the AAL0 mode.

CDS_Crc10 **Crc10 Error** **9**

In the transmit direction, this bit is set by the APU to tell the ACI to include CRC10 in the cell. The ACI computes a CRC10 value over the first 374 bits of the cell payload and writes the value into the last 10 bits of the payload. In the receive direction, the ACI calculates a CRC10 value over the first 374 bits and compares it to the value of the last 10 bits. If they do not match, the ACI sets the CDS_Crc10 bit to inform the APU. (The ACI has no means of determining if the last 10 bits are a valid CRC10 value.) If the APU finds that the cell does not contain CRC10, it can simply ignore the CDS_Crc10 bit.

CDS_Par **Parity Error Detected** **8**

This bit indicates when a parity error is detected in an incoming cell. CDS_Par is set whenever a Utopia parity error occurs, provided the global Utopia parity enable is set. The ACI_Parity bit in the ACI_Ctrl register enables/disables Utopia parity.

If the CDS_Par bit is set in the Cell Descriptor, the errored cell is placed in the ACI Error FIFO which removes the cell from the data path. Then the APU decides what action to take. If the ACI_Parity bit is cleared, then Utopia parity is ignored, the CDS_Par bit is not set, and the received cells are placed in the Receive FIFO.

The CDS_Par bit is also set by the ACI when the received cell has an HEC error and the ACI_HEC bit is set. Note that, even in case of HEC or CRC10 errors, a cell is always built in the ACI Receive FIFO. The APU may decide to discard the cell if necessary.

The table below summarizes what happens for each type of receive error. It also shows the states of the status bits in which FIFO the ACI places the errored cell for a given error type.

Error	Status Bit	Destination FIFO
HEC[1]	CDS_Par = 1	Receive FIFO
Parity[2]	CDS_Par = 1	Error FIFO
Short cell	CDS_Len = 1	Error FIFO
CRC10	CDS_Crc10 = 1	Receive FIFO
Tx Time-out	CDS_Par = 0 CDS_Len = 0	Error FIFO

1. If ACI_HEC is set, otherwise HEC is not checked
2. If ACI_Parity is set, otherwise parity is not checked.

CDS_BOM and CDS_EOM

Begin/End of Message **[7:6]**

The EDMA uses these two bits as described in Section 5.7, “AAL0 Mode Operation.” In the transmit direction, the EDMA sets the CDS_BOM bit for the first cell of a CS-PDU and the CDS_EOM bit for the last cell of a CS-PDU. Single-cell CS-PDUs have both bits set. In the receive direction and in the AAL0 mode, the EDMA uses the CDS_EOM bit to determine if the current cell is the last cell of a CS-PDU. In the receive direction, the CDS_EOM

bit is ignored in AAL5 mode and the CDS_BOM bit is ignored in both modes.

CDS_Len **Cell Length Error** **5**

This bit is set to indicate that a cell received from the Utopia PHY port has a cell length error. This error condition is generated when a short cell is received by the Utopia interface.

A short cell error is triggered if a PHY device asserts RxSOC early (that is, before the current cell is completely assembled). The partially assembled cell is placed in the ACI Error FIFO, which removes the cell from the data path. The APU can then decide what action to take but, at a minimum, should return the cell to a free list. If the free list was empty when the short cell occurred, the short cell is discarded by the ACI and the current cell is reused.

CDS_PHY[4:0]

Physical Port **[4:0]**

The field specifies the PHY port in the Utopia Master mode. In the transmit direction, the APU or the EDMA places the address of the transmit port in the CDS_PHY field. In the receive direction, the ACI Receiver places the receive port address in the CDS_PHY field.

6.4 Memory-Mapped ACI Registers

The memory-mapped registers listed in Table 6.2 provide the mechanism the APU uses for controlling ACI operations. The APU accesses these registers at memory address 0xB800.01XX, where XX is specified in the Offset column.

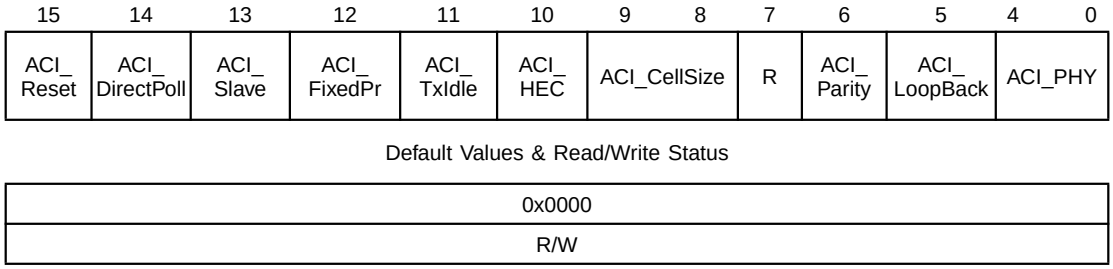
Table 6.2 Memory Mapped ACI Registers

Name	Offset	Size	R/W	Description	Initialized
ACI_Ctrl	0x00	16	R/W	ACI control field	Yes
ACI_FreeList	0x02	16	R/W	Beginning of free cell list	Yes
ACI_TxTimer	0x04	8	R/W	Transmit time-out	Yes
ACI_TxSize	0x05	8	R/W	Maximum number of cells in Transmit FIFO	Yes
ACI_TxLimit	0x06	8	R/W	Number of cells in Transmit FIFO to generate an interrupt	Yes
ACI_RxLimit	0x07	8	R/W	Number of cells in Receive FIFO to generate an interrupt	Yes
ACI_RxMask	0x08	24	R/W	Receive polling mask	Yes
ACI_Free	0x0C	32	R/W	Get or return a free cell location	–
ACI_RxRead	0x10	32	R	Get cell from Receive FIFO.	–
ACI_TxWrite	0x14	32	W	Put cell in Transmit FIFO.	–
ACI_RxCells	0x18	8	R	Number of cells in the Receive FIFO	–
ACI_TxCells	0x1A	8	R	Number of cells in the Transmit FIFO	–
ACI_Error	0x1C	32	R	Get a cell from the Error FIFO	–
ACI_RxSize	0x20	8	R/W	Maximum number of cells in Receive FIFO	Yes
ACI_BadHEC	0x26	16	R/W	Bad HEC register	–
ACI_ClearBytes	0x2B	8	R/W	ACI will use this for PAD byte	–
ACI_FreeCount	0x2F	8	R/W	Count of Free Cells	Yes

6.4.1 ACI_Ctrl Register

Figure 6.4 shows the format of the ACI_Ctrl register. Its address is 0xB800.0100. The text that follows Figure 6.4 defines the individual bits and fields in the register. The register is located at address 0xB800.0100.

Figure 6.4 ACI_Ctrl Register



ACI_Reset **Reset Transmitter and Receiver** **15**

This bit is set by PCI_RSTn (see Section 3.2, "PCI Interface") or the APU_Reset bit in the XPP_Ctrl register (page 9-22). When this bit is set, the ACI Transmitter and Receiver state machines revert to the idle state. This reset provides a mechanism for recovering from unforeseen events on the Utopia Bus. The reset does not clear the Tx and Rx FIFOs.

The reset is asynchronous and does not rely on any prerequisite conditions or clock transition. Writeable memory-mapped registers are not reset but the current state machine process is aborted asynchronously. If this reset is triggered during system run time, then the APU must reinitialize other related functional units (such as, the free list and the FIFO counters).

ACI_DirectPoll

Polling Scheme **14**

This bit specifies whether a direct or multiplexed polling scheme is used. If this bit is set, the ACI uses a direct polling method with four Clav signals and support for four slave devices. When this bit is cleared, devices 0 through 23 are polled using Clav[0] as the status line. Note that the maximum number of slave devices is 24, and that the ACI does **not** support multiplexed polling on four Clav lines.

ACI_Slave **Utopia Bus Master/Slave** **13**

This bit controls the master/slave operation of the Utopia Bus. If this bit is cleared, the ACI is the Utopia Bus master and it performs all the polling of slave devices. If this bit is set, the ACI is a Utopia slave and it responds to the ACI physical address (ACI_PHY).

Note: The ACI_LoopBack bit must never be set when the ACI_Slave bit is set.

In the slave mode, the ACI ignores the CDS_PHY address of the Cell Descriptor and, in the Master mode, it ignores the ACI_PHY field. In the Master mode, the ATMizer II+ chip emulates an ATM-layer device and in slave mode it emulates a PHY-layer device, as defined in *The ATM Forum Utopia Level 2, v1.0* specification.

ACI_FixedPr **Priority Scheme** **12**

This bit specifies the priority scheme used in the receive direction. When this bit is set, the ACI Receiver uses a fixed priority scheme, where port 0 has the highest priority and port 23 has the lowest. When this bit is cleared, a round-robin priority scheme is used.

ACI_TxIdle **Idle Cells** **11**

This bit specifies whether the ACI generates idle cells. If this bit is set, the ACI Transmitter generates idle cells whenever the Transmit FIFO is empty. The transmitter always retrieves the idle cell from cell location 0 and it always sends the idle cell to PHY address 0.

ACI_HEC **HEC** **10**

This bit specifies whether the ACI should generate or verify the HEC bit. When this bit is set, the ACI inserts the HEC byte after the cell header in the transmit direction. In the receive direction, it computes the HEC and compares it with the HEC extracted from the cell. The CDS_Par bit in the Cell Descriptor (page 6-5) is set in case of errors and the cell is placed in the ACI Receive FIFO as usual.

ACI_CellSize[1:0]

Cell Size **[9:8]**

These bits specify the cell size to be used as follows:

ACI_CellSize[1:0]	No. of Bytes in Cell
0b00	52/53
0b01	56/57
0b10	60/61
0b11	64/65

R **Reserved** **7**

Not used in the L64364.

ACI_Parity **Parity** **6**

When set, this bit specifies a global enable for Utopia parity generation and error detection. Cells with parity errors are placed in the ACI Error FIFO.

ACI_Loopback

Loopback Enable **5**

This bit enables/disables loopback operation. When this bit is set, the ACI Transmitter output is connected to the ACI Receiver input and causes the outgoing cells to loopback from the transmitter to the receiver. The Utopia Bus signals are placed in the 3-state mode.

Note: The `ACI_LoopBack` bit must never be set when the `ACI_Slave` bit is set.

ACI_PHY[4:0] **Physical Address** **[4:0]**

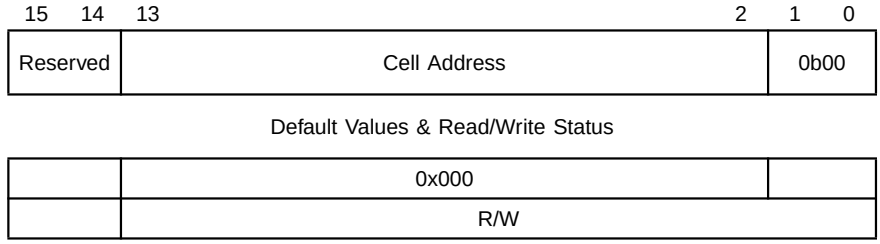
This field contains the PHY address of the ACI. In slave mode, the ACI compares incoming addresses with this field to determine if it is being addressed. This field is used only in slave mode.

6.4.2 ACI_FreeList Register

The APU uses this register only at initialization to set the beginning of the free cell list. Since cells must be aligned on a word boundary in Cell Buffer Memory, the two LSBs of the cell address are always 0b00. The ACI discards the two LSBs when the APU performs a write operation and clears the two LSBs when the APU reads the register as shown in Figure 6.5.

This register must be initialized to the first cell in the free list before the ACI_Reset bit in the ACI_Ctrl register is cleared. You can read this register during normal operation to get a snapshot of the top of the free list, but you must not write to this register again during normal operation. Its address is 0xB800.0102.

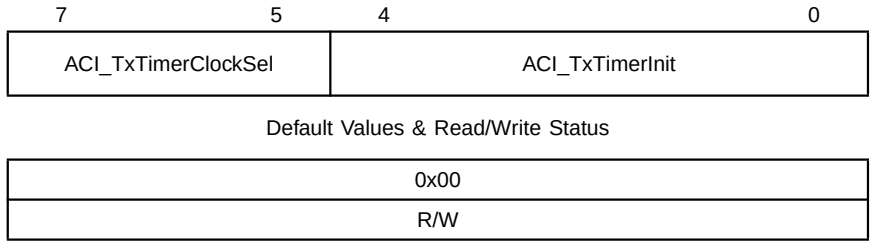
Figure 6.5 ACI_Free List Register



6.4.3 ACI_TxTimer Register

The format of the ACI_TxTimer register is shown in Figure 6.6. Its address is 0xB800.0104.

Figure 6.6 ACI_TxTimer Register Format



ACI_TxTimerClockSel

Select Timer Clock [7:5]

This field selects a clock to use for decrementing the timer. Value 0b000 specifies the system clock; values between 0b001 and 0b111 specify the time-out event of the corresponding general-purpose timer.

ACI_TxTimerInit

Timer Initialization Value [4:0]

This field specifies the timer initialization value. If this field is 0x00, the ACI Transmit Timer does not operate (that is, cells are never removed from the Transmit FIFO).

6.4.4 ACI_TxSize Register

The APU accesses this register to set the maximum size of the Transmit FIFO. This is done to guarantee sufficient free cell locations for the Receive FIFO. This register is 8-bits wide and is located at address 0B800.0105. It defaults to 0x00 at reset.

6.4.5 ACI_TxLimit and ACI_RxLimit Registers

The values in these registers set the thresholds for generating an interrupt to the APU based on the number of cells in the Transmit and Receive FIFOs. When the actual number of cells exceeds the ACI_RxLimit or drops below ACI_TxLimit, an interrupt is sent to the APU (if interrupts are enabled). These registers are each 8-bits wide and are located at addresses 0B800.0106 (Tx) and 0x0B800.0107 (Rx). Both registers default to 0x00 at reset.

6.4.6 ACI_RxMask Register

Bits 0 to 23 are assigned to PHY devices 0 to 23. When a bit is set, the corresponding PHY's CLAV (Cell Available) information is used. When a bit is cleared, the PHY's CLAV information is ignored. See Section 6.6.3, "Receive Priority Scheme," and Section 6.8, "Polling Scheme," for information about polling. This register is 24-bits wide and is located at address 0B800.0108. It defaults to 0x00 at reset.

6.4.7 ACI_Free Register

The APU accesses this register to get a free cell location (when reading) or to return a cell location to the Free Cell List (when writing). The upper 16 bits are set to 0xB000 when a cell address is returned. This register is 32-bits wide and is located at address 0B800.010C.

The following read operations from this register will cause the OCA Bus to stall or time-out:

1. A nonword (32 bit) read.
2. Any type of read with the ACI_Reset bit in the ACI_Ctrl register set.
3. Any type of read with the APU_Reset bit in the APU_AddrMap register set.

The following write operations to this register will complete, but they will be ignored and will not result in any data being written:

1. A nonword (32 bit) write.
2. Any type of write with the ACI_Reset bit in the ACI_Ctrl register set.
3. Any type of write with the APU_Reset bit in the APU_AddrMap register set.

6.4.8 ACI_RxRead Register

The APU accesses this register to retrieve a received cell from the Receive FIFO. The register is read-only. This register is 32-bits wide and is located at address 0B800.0110. The upper 16 bits are set to 0xB000 when a cell address is returned.

The following read operations from this register will cause the OCA Bus to stall or time-out:

1. A nonword (32 bit) read.
2. Any type of read with the ACI_Reset bit in the ACI_Ctrl register set.
3. Any type of read with the APU_Reset bit in the APU_AddrMap register set.

6.4.9 ACI_TxWrite Register

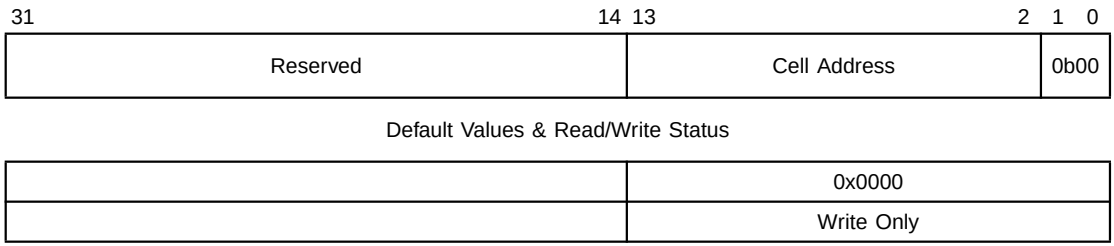
The APU writes to this register to place a cell in the Transmit FIFO. The APU may achieve the same result by issuing a Txcell command with a null Connection Number to the EDMA. Since cells must be aligned on a word boundary in Cell Buffer Memory, the 2 LSBs of the cell address are always 0b00. The ACI discards the 2 LSBs and ignores the upper 18 bits when the APU performs a write operation as shown in Figure 6.7.

The following write operations to this register will complete, but they will be ignored and will not result in any data being written:

1. A nonword (32 bit) write.
2. Any type of write with the ACI_Reset bit in the ACI_Ctrl register set.
3. Any type of write with the APU_Reset bit in the APU_AddrMap register set.

The ACI_TxWrite register is located at address 0xB800.0114.

Figure 6.7 ACI_TxWrite Register



6.4.10 ACI_RxCells and ACI_TxCells Registers

The APU reads these registers to check how many cells are currently present in the Transmit and Receive FIFOs. These registers are read-only, are each 8-bits wide, and are located at addresses 0xB800.0118 (Rx) and 0xB800.011A (Tx).

6.4.11 ACI_Error Register

The APU uses this register to retrieve cells from the Error FIFO. A read operation from this register returns a cell address (or zero if the Error FIFO is empty) and removes the cell from the Error FIFO. Note that cells may be placed in the ACI Error FIFO in case of parity errors, short cells, or transmit time-outs. This register is read only, is 32 bits wide, and is located at address 0xB800.011C. The upper 16 bits are set to 0xB000 when a cell address is returned.

The following read operations from this register will cause the OCA Bus to stall or time-out:

1. A nonword (32 bit) read.
2. Any type of read with the `ACI_Reset` bit in the ACI_Ctrl register set.
3. Any type of read with the `APU_Reset` bit in the APU_AddrMap register set.

6.4.12 ACI_RxSize Register

The APU accesses this register to set the maximum size of the Receive FIFO. This is done to guarantee sufficient free cell locations for the Transmit FIFO. The ACI Receiver stops requesting cells from the free list when the number of cells in the RxFIFO ≥ RxSize. The receive logic may have prefetched a cell before this condition was met, so it's possible to see RxSize + 1 cells in the RxFIFO.

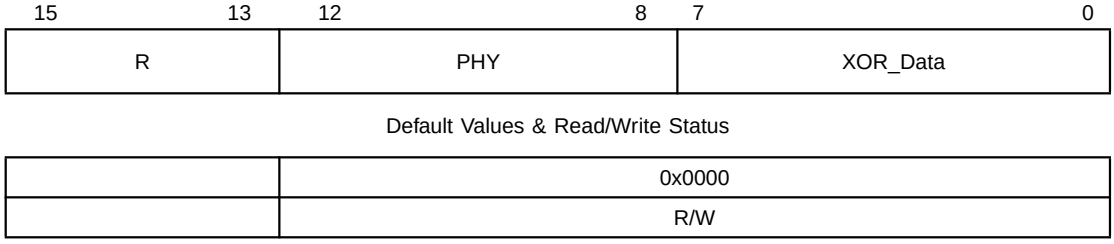
While the last prefetched cell is being received and after it goes into the RxFIFO, the ATMizer II+ stops polling (Master mode) until the FIFO count drops below RxSize. In slave mode, the CLAV signal is driven inactive until free cells are available.

The default value for this register is 0xFF (256). This value is higher than the largest possible number of free cells, which means the default operation is that the Receiver can keep fetching cells until the free list is empty. This register is R/W, is 8-bits wide, and is located at address 0xB800.0120.

6.4.13 ACI_BadHEC Register

This register is used for diagnostic purposes. The ACI uses this register to generate bad HEC for the PHY selected in the register. Figure 6.8 shows the contents of the register. It is located at address 0xB800.0126.

Figure 6.8 ACI_BadHEC Register



R **Reserved** **[15:13]**
Not used in the L64364.

PHY[4:0] **PHY Select** **R/W [12:8]**
This field specifies the PHY for which the bad HEC should be generated.

XOR_Data[7:0]

XOR Data for Generating Bad HEC **R/W [7:0]**

The data in this field is XORed with the HEC to generate a bad HEC for the cell.

When the ACI transmits a HEC for the selected PHY, it is XORed with the XOR_Data field. You can set one or more bits in the XOR_Data field to generate single or multiple bit errors for software testing. This register defaults to zero, so no bad HECs is the default operation. An illegal or unused PHY address in the PHY field or all zeros in the XOR_Data field results in this feature being disabled. See Section 6.7.5, "HEC Generation," for a more complete description of the ACI_BadHEC register.

6.4.14 ACI_ClearBytes Register

The ACI uses the pattern given in this register to pad the bytes remaining in a cell after CDS_TBytes. The remaining bytes of the payload (48 CDS_Tbytes) are not read from Cell Buffer Memory. Instead, they are replaced by a constant value equal to the value of the ACI_ClearBytes register. This is normally used in AAL0 mode only. See also Section 6.3, "Cell Descriptor." The ACI_ClearBytes register is R/W, 8-bits wide, located at 0xB800.012B, and defaults to a value of 0x00.

6.4.15 ACI_FreeCount Register

This register is a software debugging aid. The APU can use this register to keep track of the number of free cells in CBM. This register is incremented and decremented by the ACI as cells are added to and removed from the free list. It is up to you to initialize this register with a valid number. This register is R/W, 8-bits wide, located at address 0B800.012F, and defaults to a value of 0x00.

6.5 Cell Buffer Manager

The Cell Buffer Manager manages the Free Cell List, the Transmit FIFO, the Receive FIFO, and the Error FIFO.

The Cell Buffer Manager maintains the Free Cell List using a hardware register called the ACI_FreeList register. At initialization, the APU must

build a list of free cells by executing a routine similar to that shown in Section 6.5.1, “Cell Buffer Initialization.”

Cell number 0 is reserved and must never be used except for storing the idle cell pattern. If the idle cell generation is inhibited, cell location 0 may be used as scratch-pad APU memory.

6.5.1 Cell Buffer Initialization

The following C code performs the initialization:

```
typedef struct Cell_s {
        ushort              CDS_Next;
        ushort              CDS_Tbytes:6,
                            CDS_Crc10:1,
                            CDS_Par:1,
                            CDS_BOM:1,
                            CDS_EOM:1,
                            CDS_Len:1,
                            CDS_PHY:5;
#ifdef CELL_TAG
        ulong               Tag[CELL_TAG];
#endif
        ulong               CellHdr;
        uchar               Payld[48];
} Cell_t *pCell_t;

Cell_t                      CellBuff[BufferSize];
ushort                      TxHead, TxTail;
ushort                      RxHead, RxTail;
ushort                      FreeList;

for (ushort i = 1; i < BufferSize - 1; i++)
        CellBuff[i].Next = (i + 1) * sizeof(Cell_t);
CellBuff[BufferSize - 1].Next = 0;
FreeList = 1;
```

6.5.2 Requesting and Releasing a Free Cell Location

When the APU or the ACI Receiver requests a free cell location, the Cell Buffer Manager returns the ACI_Free register and reloads the register with the CDS_Next field of the current Cell Descriptor. If the Free Cell List

is empty, the Cell Buffer Manager returns the value zero. This is illustrated by the following C routine. Note that this routine and the one in Section 6.5.3, “Inserting and Removing Cells from the ACI FIFO,” are executed by the Cell Buffer Manager hardware unit and not by the APU.

```
1. ulong GetFree()
2. {
3.        ulong CellNum = FreeList;
4.        if (FreeList)
5.              FreeList = CellBuff[CellNum].Next;
6.        return CellNum;
7. }
```

When a cell is returned to the Free Cell List the Cell Buffer Manager executes the following routine:

```
1. void ReturnFree(ulong CellNum)
2. {
3.        CellBuff[CellNum].Next = FreeList;
4.        FreeList = CellNum;
5. }
```

6.5.3 Inserting and Removing Cells from the ACI FIFO

Four registers maintain the Transmit and Receive FIFOs. They are called the ACI_TxHead, ACI_TxTail, ACI_RxHead, and ACI_RxTail registers. All four registers are cleared at initialization. The management of the Transmit and Receive FIFOs is identical.

When a cell needs to be put into a FIFO, the Cell Buffer Manager executes the following routine:

```
1. void Put(ulong CellNum)
2. {
3.        if (Tail != 0)
4.              CellBuff[Tail].Next = CellNum;
5.        else
6.              Head = Tail = CellNum;
7.        CellBuff[CellNum].Next = 0;
8. }
```

When a cell needs to be removed from a FIFO, the Cell Buffer manager executes the following routine:

```
1. ulong Get()
2. {
3.        ushort CellNum = Head;
4.        if (Head)
5.              Head = CellBuff[Head].Next;
6.        if (Head == 0)
7.              Tail = 0;
8.        return CellNum;
9. }
```

6.5.4 Setting and Checking FIFO Sizes

Since the APU creates the Free Cell List at initialization, it may decide to use only part of the Cell Buffer Memory for cell holders and leave the rest of it as a scratch-pad memory. The size of both FIFOs is therefore implicitly set during the initialization as described in Section 6.5.1, “Cell Buffer Initialization.”

Since both FIFOs share the same memory, it may be necessary to limit the size of the Transmit FIFO to assure a sufficient size for the Receive FIFO. This can be done by loading the desired maximum size of the Transmit FIFO into the ACI_TxSize register.

The ACI uses the ACI_TxCells and ACI_RxCells registers to maintain a count of the number of cells in the Receive and Transmit FIFOs. When a cell is inserted into a FIFO, the ACI increments the corresponding counter; when a cell is removed, the counter is decremented. If the ACI_TxCells counter reaches a count equal to the value in the ACI_TxSize register, the Cell Buffer Manager returns Cell Number 0 the next time the APU requests a free cell location. Due to the presence of the EDMA Request Queue, the Transmit FIFO size can exceed the value programmed in the ACI_TxSize register by a maximum of four cells.

6.6 ACI Receiver

The ACI Receiver accepts bytes of cell data from the Utopia Receive Bus and builds cells in Cell Buffer Memory. When a cell arrives, the APU reads the cell header to determine the Virtual Connection to which the cell belongs and what operations to perform.

The received cells can be handled in one of four ways:

1. **Reassembly**–the APU issues an RxCell command to the EDMA to transfer the cell into a buffer.
2. **Cell buffer switching**–the APU sends the received cell back to the PHY device (either the same one that sent it or a different one) without transferring the cell to external memory.
3. **External memory switching**–the APU issues a command to transfer the whole cell (including the header) to external memory.
4. **Discard**–the APU discards the cell and returns the cell number to the Free Cell List.

6.6.1 ACI Receiver Operations

When the cell arrives from the Utopia Receive Bus, the ACI Receiver requests a free cell location from the Cell Buffer Manager. The Cell Buffer Manager gets a cell location from the Free Cell List. In case there are no more free locations, the ACI Receiver stops cell reception. When the cell is completely built in Cell Buffer Memory, the ACI Receiver signals the Cell Buffer Manager to place the cell in the Receive FIFO.

The APU retrieves the cell address from the Receive FIFO by reading the ACI_RxRead register and the Cell Buffer Manager removes the cell from the Receive FIFO. The APU then reads the cell header to determine what operations to perform on the cell. If reassembly into a buffer is required, the APU issues the RxCell command by writing to the EDMA_RxCell register, specifying the Connection Number and the Cell Number.

When the EDMA completes a cell transfer and the VCD_CellHold bit in the VC Descriptor (page 5-11) is cleared, it sends the cell address to the Cell Buffer Manager to place the cell location in the Free Cell List. If the VCD_CellHold bit is set, the APU is responsible for returning the cell to the Free List by writing the cell address into the ACI_Free register.

If the APU decides to send the cell back to the Utopia interface (internal switching), possibly after modifying the cell header or payload, it places the cell in the Transmit FIFO by writing the cell address into the ACI_TxWrite register. Or, the APU issues a TxCell command with a null Connection Number and the EDMA places the cell in the Transmit FIFO. This latter method maintains the transmit cell sequence. Then the Cell Buffer Manager updates the Transmit FIFO pointers.

Finally, the APU can choose to discard the cell by writing the cell address to the ACI_Free register.

6.6.2 Receive FIFO Status

The APU has several ways to verify the status of the ACI Receive FIFO.

- Reading the ACI_RxCells register (page 6-16) returns the number of cells in the ACI Receive FIFO.
- Reading the ACI_RxRead register (page 6-15) returns a cell address (and removes the cell from the FIFO) or returns a value of 0 if the ACI Receive FIFO is empty.
- The `APU_ACI_RxFull` bit in the APU_Status register (page 4-108) is set when the ACI Receive FIFO is full. The FIFO is full when the number of cells in the FIFO exceeds the value programmed in the ACI_RxSize register or if all cells in CBM are used up. If the corresponding interrupt is enabled, this condition generates the `IntACI_RxFull` vectored interrupt.

  Note: This bit is also set when the ACI_RxSize register (page 6-17) and the ACI_RxCells register (page 6-16) are both zero.

- The `APU_ACI_RxThrld` bit in the APU_Status register is set when the number of cells in the ACI Receive FIFO exceeds the value programmed in the ACI_RxLimit register (page 6-14). If the corresponding interrupt is enabled, this condition generates the `IntACI_RxThrld` vectored interrupt.

6.6.3 Receive Priority Scheme

When operating in either a multi-PHY environment or as the Utopia master, there can be more than one physical device ready to send a cell at any given time. All PHY devices are polled for cell availability according to Utopia Level 2 protocol. The ACI Receiver then decides which port to service based on the port number and the programmed port priority scheme. The ACI performs port polling for the next cell slot during the current cell transfer. This overlapping of polling and data transfers makes it possible for the ACI to receive cells back-to-back.

The `ACI_FixedPr` bit in the ACI_Ctrl register (page 6-10), determines the port priority scheme that is used. If this bit is set, the ports are serviced

with a fixed priority where port 0 has the highest priority and port 23 the lowest. In a given polling cycle, if port 0 signals it has a cell to send, it will always be selected. If port 1 signals it has a cell to send, it will be selected only if port 0 does not have a cell to send.

When the ACI_FixedPr bit is cleared, a round-robin priority scheme is used. For the first cell after reset, port 0 has the highest priority and port 23 the lowest. After a port receives service, it then becomes the lowest priority port and the next port immediately following it (in descending order with wrap around from 23 to 0) becomes the highest priority. This process continues indefinitely or until the ACI_FixedPr bit changes.

6.6.4 HEC Processing

The ACI may be programmed to receive cells with or without HEC bytes. The ACI_Ctrl register (page 6-10) includes a ACI_HEC control bit. If the ACI_HEC bit is set, the ACI expects the received cells to have a HEC byte.

Cells containing a HEC byte have an actual cell size on the Utopia Bus of 53-, 57-, 61-, or 65-bytes depending on the state of the ACI_CellSize field in the ACI_Ctrl register.

The first step in HEC processing involves comparing the received cell's HEC byte with a computed HEC byte. If an error is detected, the CDS_PAR bit in the Cell Descriptor (page 6-5) is set. Despite the error, the cell is always received and placed in the Receive Cell Buffer. Then the APU has the responsibility of deciding whether to retain or discard a cell that contains a HEC error.

6.6.5 CRC10 Verifications

The ACI Receiver systematically checks the CRC10 of the cell although it has no means to detect if the cell actually contains a CRC10. The result of the check sets or clears the CDS_Crc10 bit in the Cell Descriptor (page 6-5). In most cases (except for AAL3/4 traffic) the CDS_Crc10 bit is set because the cell did not contain CRC10 at all. The decision to exploit this error indicator or not is left to the APU and the cell is always built in the Receive FIFO.

6.6.6 Utopia Parity Checking

The ACI may be programmed to receive cells with or without Utopia parity on a global basis. The ACI_Parity bit in the ACI_Ctrl register (page 6-10) is used to enable this function.

Parity checking is performed on a per octet basis. If an error is detected, the cell is placed in the Error FIFO and the CDS_Par bit in the Cell Descriptor (page 6-5) is set. This capability makes it possible to detect contention on the Utopia Bus for multi-PHY applications.

6.7 ACI Transmitter

The ACI Transmitter handles cell transfers from the Transmit Cell Buffer Memory to the Utopia Transmit Bus, one byte at a time. There are three possible ways for cells to become available for transmission in Cell Buffer Memory.

1. **Segmentation**–the APU issues a TxCell command to the EDMA to build an AAL5 cell in Cell Buffer Memory. After the cell is built, if CellHold is cleared, the CBM places the cell in the Transmit FIFO.
2. **Internal switching**–without using external memory, the ACI Transmitter outputs cells to the Utopia Transmit Bus that were previously received by the ACI Receiver and placed in the Receive Cell Buffer Memory.
3. **External memory switching**–transmits cells that were brought into the Cell Buffer Memory from external memory in AAL0 mode.

6.7.1 ACI Transmitter Operations

The APU decides which Virtual Connection should be serviced at any given time. After the APU makes that decision, it requests a free cell location from the Cell Buffer Manager by reading the ACI_Free register. If there are no free cells or if the number of cells in the Transmit FIFO has reached the maximum value programmed in the ACI_TxSize register, the Cell Buffer Manager returns a value of 0. Otherwise, it returns a valid cell address. The APU then issues a Txcell command which specifies the Connection Number and the cell address.

The EDMA uses the cell address to build a cell in the Cell Buffer. After the cell is built, if the VCD_CellHold bit is clear, the EDMA sends the cell address to the Cell Buffer Manager. The Cell Buffer Manager then places the Cell in the Transmit FIFO. If the VCD_CellHold bit is set, only the APU can send the cell address to the Cell Buffer Manager.

The APU can also build the cell using the cell address obtained from the Cell Buffer Manager. In addition, the APU can place the newly built cell in the Transmit FIFO either directly by writing into the ACI_TxWrite register or indirectly by executing a Txcell command with a null Connection Number.

The ACI Transmitter systematically retrieves cells from the Transmit FIFO. If the Transmit FIFO is empty, the ACI Transmitter can optionally send idle cells. If the Cell Buffer Manager provides a valid address, the ACI Transmitter sends the cell, one byte at a time, to the Utopia interface. After the cell is transmitted, the ACI Transmitter tells the Cell Buffer Manager to return the cell to the Free Cell List.

6.7.2 Transmit FIFO Status

The APU has several ways to verify the status of the ACI Transmit FIFO.

- Reading the ACI_TxCells register returns the number of cells in the ACI Transmit FIFO.
- Reading the ACI_Free register returns a free cell location address (and removes the cell from the Free Cell List) or, if the ACI Transmit FIFO is full, it returns a value 0. The FIFO is considered full when the number of cells in the FIFO exceeds the value programmed in the ACI_TxSize register.
- The APU_ACI_TxThrld bit in the APU_Status register (page 4-108) is set when the number of cells in the ACI Transmit FIFO is less than the value programmed in the ACI_TxLimit register. If the corresponding interrupt is enabled, this condition generates the IntACI_TxThrld vectored interrupt.

6.7.3 Idle Cell Generation

If the ACI_TxIdle bit in the ACI_Ctrl register (page 6-10) is set when the Transmit FIFO becomes empty, the ACI Transmitter retrieves the idle cell from Cell Number 0 and sends it to the Utopia Transmit Bus. The APU

must have previously initialized Cell Number 0 with the appropriate bit pattern to create an idle cell. Idle cells are always transmitted to the PHY port 0.

If the ACI_TxIdle bit is clear when the Transmit FIFO becomes empty, the ACI Transmitter stops cell transmission. Transmission resumes only after the Transmit FIFO becomes nonempty.

Idle cell generation is intended for single PHY applications, and the PHY port must be mapped to location 0. In a multi-PHY environment, it is possible to send idle cells to address 0 only. The other PHY addresses must use PHY devices that are capable of generating the idle cells.

6.7.4 PHY Port Selection and Port Polling

In the Utopia Master mode, the PHY address in the transmit direction is stored in the VCD_PHY field of the VC Descriptor (page 5-11). The EDMA transfers that field into the CDS_PHY field of the Cell Descriptor (page 6-5). Based on the contents of the CDS_PHY field, the ACI Transmitter selects the port to which it will send the data. Then the ACI Transmitter polls the selected device to see if it can accept data. If the PHY port is not ready to accept the data, the ACI Transmitter stalls until the PHY device becomes ready.

This mechanism can cause head-of-the-line blocking in a multi-PHY environment. To avoid this situation, use careful cell scheduling so that long bursts to the same PHY device do not occur.

A timer is used to detect situations where the destination PHY device is malfunctioning. Upon time-out, the cell at the head of the transmit FIFO optionally can be discarded.

6.7.5 HEC Generation

When the ACI_HEC bit in the ACI_Ctrl register (page 6-10) is set, the ACI Transmitter generates the HEC byte and inserts it after the cell header. Depending on the state of the ACI_CellSize bits, the ACI Transmitter outputs 53-, 57-, 61-, or 65-bytes per cell.

When the ACI_HEC bit is cleared, the ACI Transmitter does not generate an HEC byte and, depending on the state of the ACI_CellSize bits, the ACI Transmitter outputs 52-, 56-, 60-, or 64-bytes per cell.

The ACI_BadHEC register can be used to generate bad HECs for diagnostic purposes. This feature can be targeted to a specific PHY. The format of the ACI_BadHec register is shown in Figure 6.8.

6.7.6 CRC10 Generation

When the CDS_Crc10 bit in the Cell Descriptor (page 6-5) is set, the ACI Transmitter generates 10 bits of CRC10 out of the first 374 bits of cell payload and replaces the last 10 bits of the cell payload with the calculated CRC10.

6.7.7 ACI Transmitter Time-Out

When the destination PHY device is not ready to receive a cell, the ACI Transmitter stalls. To prevent a malfunctioning PHY device from blocking other PHY devices, the cell at the head of the transmit FIFO is discarded when the ACI Transmit Timer reaches time-out.

If the ACI Transmitter stalls because a PHY device is not ready, it loads the contents of the ACI_TxTimer register (see Section 6.4.3, "ACI_TxTimer Register," page 6-13) into the ACI Transmit Timer. The Timer decrements using a system clock or a time-out event of a general-purpose timer. After the ACI Transmit Timer decrements to 0, the ACI Transmitter removes the cell at the head of the Transmit FIFO.

The ACI Transmitter places the discarded cell in the ACI Error FIFO from which the APU may retrieve it by reading the ACI_Error register. Subsequent sequential cells that are also queued for the same PHY port are also placed in the ACI Error FIFO, providing Tx_Clav (page 3-12) is not asserted.

If the ACI Error FIFO is empty when the APU reads the ACI_Error register, a value of 0 is returned. The ACI Transmitter can optionally send an interrupt whenever the ACI Error FIFO is not empty.

6.7.8 Utopia Parity Generation

The ACI Transmitter always generates a parity signal. This signal can be ignored if parity is not used.

6.8 Polling Scheme

In the multi-PHY environment, the Utopia master must poll slave devices to obtain their status. The L64364 supports two polling methods in the receive direction:

- Direct polling scheme using four Tx_Clav (page 3-12) signals
- Multiplexed polling using a single Tx_Clav line

The L64364 does not support multiplexed polling on four Tx_Clav lines.

When the ACI_DirectPoll bit in the ACI_Ctrl register (page 6-10) is set, the ACI polls up to four slave devices in parallel using the four Tx_Clav[3:0] signals. The ATMizer II+ chip assigns PHY addresses 3 to 0 to the Tx_Clav[3:0] lines respectively. This is not programmable. When the ACI_DirectPoll bit is cleared, the ACI uses the Tx_Clav[0] signal to poll PHY devices 0 through 23. In this mode, Tx_Clav[3:1] are not used.

To avoid polling nonexistent PHY devices in the receive direction, the ACI_RxMask register (page 6-14) must be correctly programmed. The ACI Receiver always polls all (4 or 24) PHY devices, however, only if bit N in the ACI_RxMask register is set will the Tx_Clav bit for PHY N be processed. If the bit is cleared, the ACI Receiver ignores that device.

6.9 Loopback Mode

The ACI includes a loopback mode which is enabled when the ACI_LoopBack bit in the ACI_Ctrl register (page 6-10) is set.

Note: The ACI_Slave bit must not be set at the same time as the ACI_LoopBack bit.

In the loopback mode, the ACI Transmitter acts as a Utopia master and the ACI Receiver acts as a Utopia slave. As a slave device, the Receiver responds to any address and ignores the value in the ACI_PHY field of the ACI_Ctrl register. The value of 0x1F is written to the CDS_PHY field of all cells received in loopback mode. The input signals use internal multiplexers to connect to the appropriate output signals. All external Utopia signals are 3-stated when in loopback mode.

6.10 Utopia Interface

Utopia Interface timing conforms to *ATM Forum Utopia Level 2, v1.0.* The L64364 ATMizer II+ chip directly connects to PHY devices (in Master mode) or ATM Layer devices (in slave mode) that conform either to the *Utopia Level 2 multi-PHY* or *Utopia Level 1* cell-level handshake specifications.

6.10.1 Utopia Clocks

The TX_CLK and RX_CLK signals can be asynchronous to each other and to the internal system clock. The frequencies of TX_CLK and RX_CLK must be no more than 80% of the frequency of the internal system clock. For example for an internal system clock at 66 MHz, TX_CLK and RX_CLK frequencies can be no greater than 50 MHz.

6.10.2 Unused Pins

The following signals are 3-stated at various times and therefore need to be pulled to their inactive states. In Slave mode, or in Master mode when multiplexed polling is used, the TX_CLAV[3:1] and RX_CLAV[3:1] lines should be tied to logic or digital ground through 4.7 kΩ resistors. The TxSOC and RxSOC signals should be tied to logic or digital ground through 4.7 kΩ resistors. The RxEnbl and TxEnbl signals should be pulled to logic 1 with resistors.

Chapter 7
Scheduler Unit

This chapter describes the Scheduler Unit and includes the following sections:

- Section 7.1, “Scheduler Overview,” page 7-1
- Section 7.2, “Priority Mode Operation,” page 7-3
- Section 7.3, “Flat Mode Operation,” page 7-8
- Section 7.4, “Calendar Switching,” page 7-11
- Section 7.5, “Command Execution,” page 7-13
- Section 7.6, “Register Descriptions,” page 7-14

Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” must not be modified.

7.1 Scheduler Overview

The Scheduler performs traffic management functions on a large number of connections with arbitrary cell rates. Traffic management involves managing a Calendar Table that is typically implemented in an external memory. The Scheduler executes the following commands when the APU accesses appropriate memory-mapped registers:

- `Schedule (ConNum, T)` – Schedules the ConNum for service at cell slot T.
- `Tic` – Advances the time to the next cell slot.
- `Service` – Returns the ConNum to be serviced in the current slot.
- `Calendar Switch (Cal_no)` – Switches the active calendar to `Cal_no`.
- `Now(T)` – Moves the current slot pointer to a different slot number T.

Each entry in the Calendar Table corresponds to one cell slot. An entry contains the Connection Numbers of the Virtual Connections (VCs) to be serviced in that slot. Since there may be more than one VC scheduled for service in one cell slot, the VC's are kept in a linked list using the `NextVCD` field of VC Descriptors (page 5-8).

Since each entry in the Calendar Table holds either one or two Connection Numbers, the required memory size is proportional to the number of bits required to encode a Connection Number. The Scheduler can also support a limited number of VC Descriptors located in Cell Buffer Memory.

The Scheduler operates in one of two modes, Priority mode or Flat mode. In Priority mode, the connections requiring service are assigned priorities based on the `Class` field of the VC Descriptor. There are six priority levels. Class 0 has the highest priority and Class 5 has the lowest. In Flat mode, all connections have equal priority and the `Class` field is not used.

When a connection is scheduled in a particular slot, it can either be inserted at the bottom of the list in a given class (tail-insertion) or it can be inserted at the top of the list in a given class (head-insertion). The selection can be made on a per-class basis. In Flat mode, since there is only one class, new connections are put either at the beginning of the list or at the end.

In Priority mode, the Calendar Table holds only the head of a list of VCs. The Scheduler must scan the list, resulting in the possibility of long execution times and many memory accesses. In Flat mode, the Calendar Table holds pointers to both the head and tail of the list which assures constant and predictable processing time.

Since retrieving a connection for service and advancing the time are separate operations, the APU can use the current cell slot for a connection that is not managed using the Scheduler. In this case, handling the priorities among connections is totally under software control.

The scheduler supports four calendars (0 to 3). All four calendars operate in either Priority or Flat mode. You can switch between calendars at any time. Once a calendar is selected, all future commands are

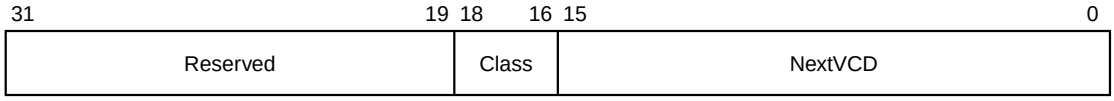
executed on that calendar until another is selected. The mechanism for switching calendars is described in Section 7.4, “Calendar Switching.”

All entries in the Calendar Table must be initialized before using the Scheduler.

7.2 Priority Mode Operation

In Priority mode, the Scheduler handles six priority classes. Class 0 has the highest priority and the Class 5 the lowest. The first word of the VC Descriptor contains the priority class and the `NextVCD` link field as shown in Figure 7.1.

Figure 7.1 VC Descriptor Format (Word 0)

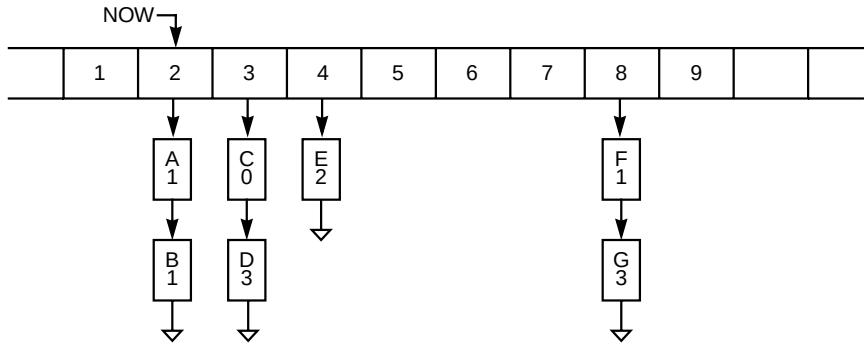


In Priority mode, each Calendar Table entry holds a pointer to the head of the VC Descriptor list that is to be serviced in the corresponding cell slot.

7.2.1 Example of Priority Mode Operation

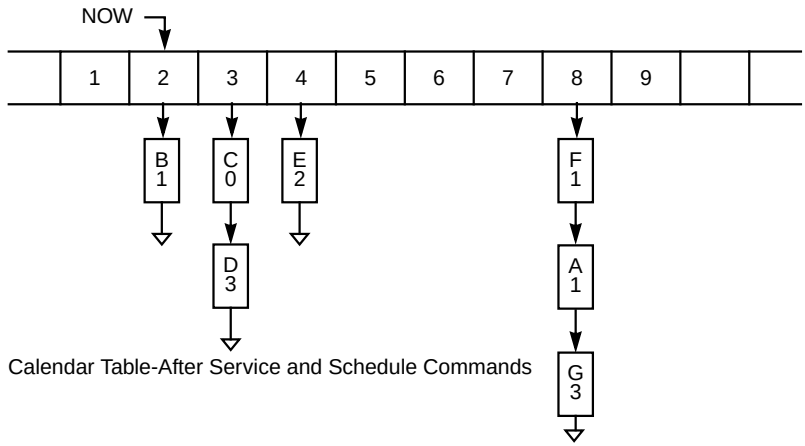
In Figure 7.2, slot 2 is the current cell slot. Slot 2 holds Connections A and B, which belong to Class 1.

Figure 7.2 Scheduler Calendar Table in Priority Mode



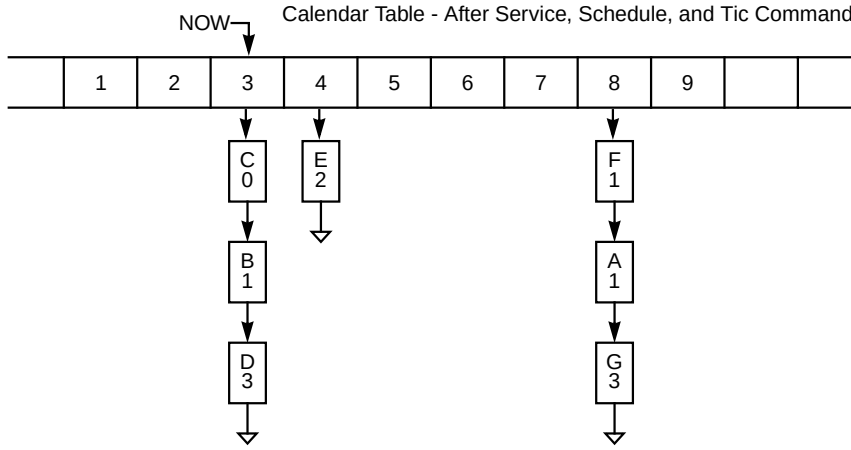
To determine which connection needs to be serviced at the current time, the APU executes a `service` command. In response to the `service` command, the Scheduler returns Connection A and then removes it from the list by advancing the head pointer to Connection B. In typical applications, after a connection is serviced, it must be rescheduled to be serviced again and the APU must issue a `schedule` command to place the VC Descriptor back on the list. For example, the intercell gap for Connection A equals 6, then Figure 7.3 depicts the Calendar Table after the completion of the `schedule` command.

Figure 7.3 Priority Mode - Calendar Table



Note that the default mode for the scheduling of a new connection in the list is tail-insertion. Whenever a new connection is scheduled, it is scheduled at the end of the corresponding class. You can optionally program the scheduler to use head-insertion for a particular class. After Connection A has been serviced and rescheduled, the current time is advanced and the APU issues the `tic` command. Then the Scheduler advances its current index and attaches the unserviced Connection B to slot 3 as shown in Figure 7.4. Since Connection B has a lower priority than Connection C, it is inserted after Connection C.

Figure 7.4 Priority Mode - Calendar Table

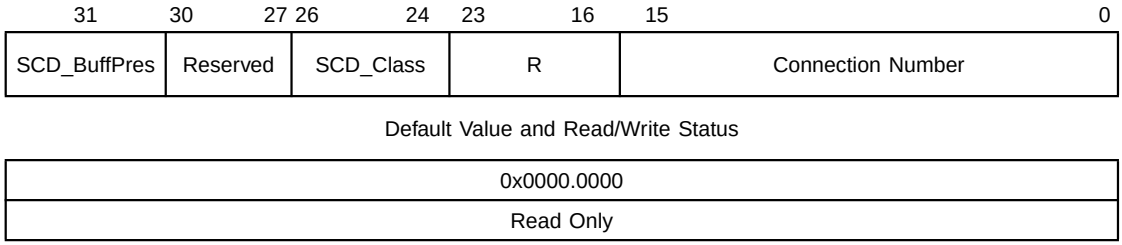


7.2.2 Service Command

In Priority mode, the Scheduler maintains the head and tail of the Connection List for each class at the current cell slot in internal registers for each calendar. This reduces the number of memory accesses. When the APU reads the SCD_Serv register, the Scheduler returns the head of the highest priority nonempty list and advances the head pointer to the next connection. If all lists are empty, the Scheduler returns Connection Number 0.

In addition, most applications need to know the connection class and whether the connection has any data to send. To support these requirements, the Scheduler returns the `SCD_BuffPres` and `SCD_Class` fields along with the `Connection Number` field as shown in Figure 7.5. It also reads the next VCD pointed to by the new head pointer to have the `VCD_BuffPres` and `VCD_Class` information ready for the next time the `service` command is issued.

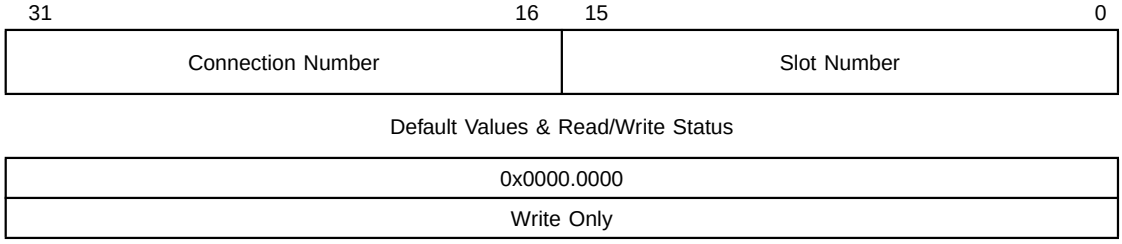
Figure 7.5 Service Command Return Value



7.2.3 Schedule Command

The Scheduler executes the schedule command when the APU writes the Connection Number and the cell slot number into the SCD_Sched register. Figure 7.6 shows the format of this register. It is located at address 0xB800.0210.

Figure 7.6 SCD_Sched Register Format



If Connection Number 0 is specified in the Connection Number field, the Scheduler uses the last non-null Connection Number of the current calendar returned by a service command. If a non-null Connection Number is specified for a schedule command, the Scheduler reads the corresponding VC Descriptor (page 7-3) to retrieve the connection class; otherwise, the class is kept in an internal register.

If the slot number in the schedule command is equal to the current time slot as pointed to by the Now register, the connection is scheduled in the next slot.

It is preferable to use a null connection number for the schedule command whenever possible because this produces fewer memory accesses.

The scheduler can also insert the new scheduled connection at the head (instead of the tail) of the same class in Priority mode. This option can be enabled for each class by setting the corresponding `Head_Sel` bit in the SCD_HeadSel register (see Figure 7.7). The SCD_HeadSel register is located at address 0xB800.0243.

Figure 7.7 Format of SCD_HeadSel Register

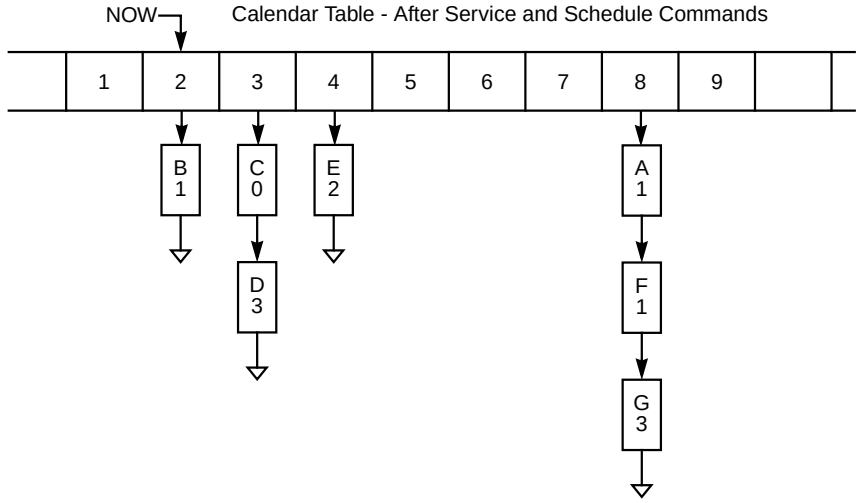
7	6	5	4	3	2	1	0
R		Head_Sel5	Head_Sel4	Head_Sel3	Head_Sel2	Head_Sel1	Head_Sel0

Default Value and Read/Write Status

0x00
R/W

Referring to Figure 7.8, if head insertion mode is chosen for Class 1, a command to Schedule connection A in slot 8 will make the Scheduler schedule connection A before F.

Figure 7.8 Priority Mode - Calendar Table



7.2.4 Tic Command

When the APU writes to the SCD_Tic register, the Scheduler executes a `tic` command. Then the Scheduler advances the internal `now` index, wrapping around the Calendar Table if needed. The Calendar end is set by the contents of the SCD_CalSize register. The Scheduler scans the list of connections present in the new cell slot and appends the list of connections to the end of the corresponding priority class list. Note that the connections from the next slot are always appended to the end of the corresponding priority class in the current list independent of the `Head_Sel` bit. A read of the SCD_Tic register returns the slot number of the current calendar.

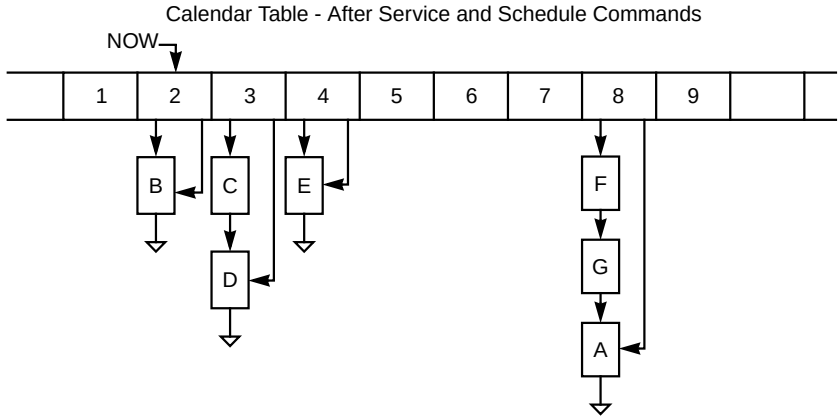
7.3 Flat Mode Operation

In Flat mode, each entry in the Calendar Table holds pointers to both the head and tail of the list of connections. A `schedule` command uses a fixed amount of time, but all connections effectively have the same priority and the `Class` field of the VC Descriptor is not used.

7.3.1 Example of Flat Mode Operation

Assuming the same initial situation as the one shown in Figure 7.2, Figure 7.9 depicts a Calendar Table after execution of the `service` and `schedule` commands.

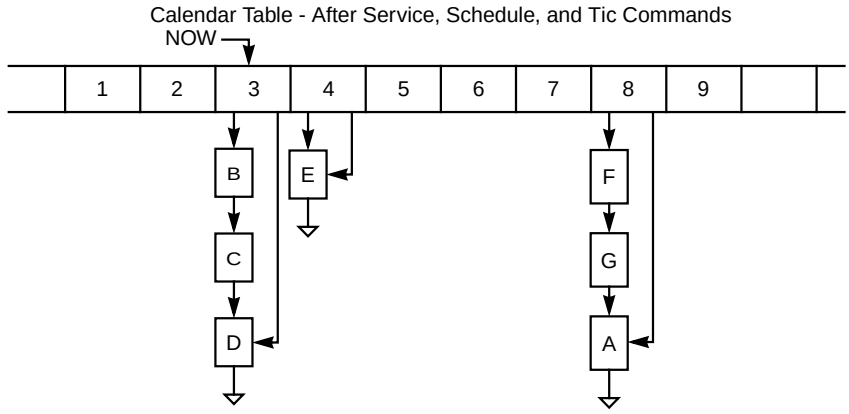
Figure 7.9 Flat Mode - Calendar Table after Schedule Command



In this case, Connection A was appended after Connection G at the end of the list in cell slot 8. Since the last connection on a list is available using the tail pointer, the Scheduler does not have to scan the connection list at cell slot 8 when executing a `schedule` command.

After Connection A is serviced and rescheduled, the current time must be advanced so the APU issues a `tic` command. The Scheduler advances its current index and attaches the unserviced Connection B to slot 3. Since there are no priorities, Connection B is attached in front of the list. Figure 7.10 shows the Calendar Table after execution of the `service`, `schedule`, and `tic` commands.

Figure 7.10 Flat Mode - Calendar Table after Tic Command



7.3.2 Service Command

When the APU reads the SCD_Serv register, the Scheduler executes the `service` command. Since head and tail pointers of the current cell slot are held in the Scheduler's internal registers, the APU read operation is completed immediately. After the read is completed, the Scheduler replaces the head pointer with the `VCD_NextVCD` field of the returned VC Descriptor.

In addition, most applications need to know whether the connection to be serviced has any data to send. To support this requirement, the Scheduler returns the `SCD_BuffPres` field along with the `Connection Number` as shown in Figure 7.5. It also reads the next VCD pointed to by the new head pointer to have the `VCD_BuffPres` and `VCD_Class` information ready for the next time the `service` command is issued.

Note: The VCD_Class field is passed on as the SCD_Class field in the return value when Flat mode is selected.

7.3.3 Schedule Command

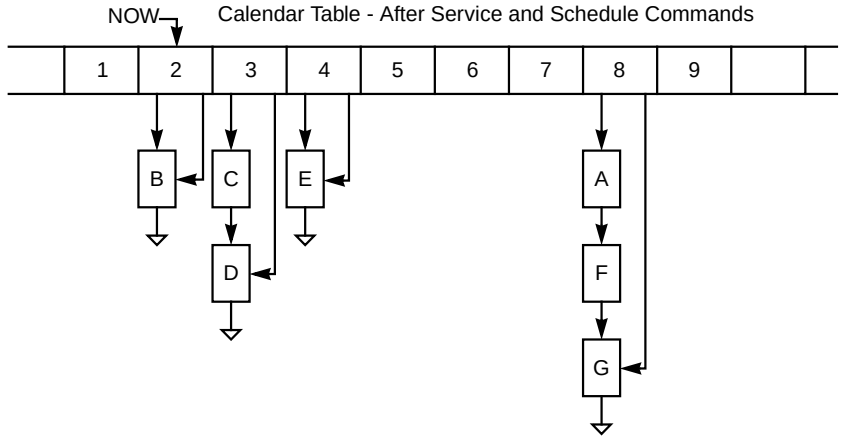
The Scheduler executes the schedule command when the APU writes the connection number and the cell slot number into the SCD_Sched register. Figure 7.6 shows the format of this register. If Connection Number 0 is specified in the Connection Number field, the Scheduler uses the last non-null connection number of the current calendar returned by a service command.

If the slot number in the schedule command is equal to the current time slot, the connection is scheduled in the next slot.

It is preferable to use a null connection number for the schedule command whenever possible because this produces fewer memory accesses.

The scheduler can also insert the new scheduled connection at the head (instead of the tail) of the list. This option can be enabled by setting the SCD_HeadSel0 bit of the SCD_HeadSel register. Referring to Figure 7.11, if head-insertion mode is chosen, a command to schedule connection A in slot 8 will make the scheduler schedule connection A before F.

Figure 7.11 Flat Mode - Calendar Table with SCD_HeadSel0 Bit Set



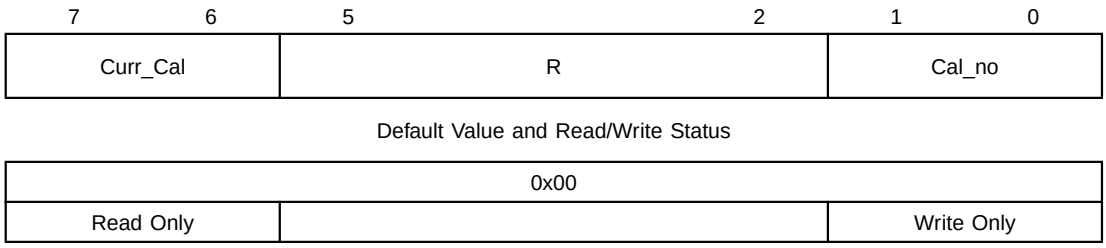
7.3.4 Tic Command

The Scheduler executes the `tic` command when the APU writes to the SCD_Tic register (page 7-14). The Scheduler advances its internal index register, wrapping around the calendar end (the calendar end is set by the contents of the SCD_CalSize register, page 7-14). The Scheduler attaches any connections scheduled for service in the next cell slot to the end of the list in the current cell slot independent of the `Head_Sel` bit. A read of the SCD_Tic register returns the current slot number of the current calendar.

7.4 Calendar Switching

Four calendar tables (0 to 3) are supported in the ATMizer II+. The `Cal_Switch` command is used to switch between calendars. The APU issues the `Cal_Switch` command by writing to the SCD_CalSwitch register with a new calendar number in the `Cal_no` field. If, however, the new calendar number is the same as the old calendar, there is no calendar switching. The format of the SCD_CalSwitch register is shown in Figure 7.12. The command only switches between calendars; it does not advance any pointers in the new or old calendar. All commands issued prior to the `Cal_Switch` command are completed before the calendar is switched. The SCD_CalSwitch register is located at address 0xB800.0223.

Figure 7.12 Format of the SCD_CalSwitch Register



Curr_Cal[1:0] Current Calendar Number [7:6]
This field can be read any time to return the current calendar number.

R Reserved [5:2]
Not used in the L64364.

Cal_no[1:0] **New Calendar Number** **[1:0]**
This field contains the number of the calendar to which the `Cal_Switch` command switches.

When the APU issues the `Cal_Switch` command with the new calendar number (`Cal_no` = 0b00 to 0b11), the Scheduler switches to the new calendar and the NOW pointer points to the new calendar. For example, Figure 7.13 shows the NOW pointer pointing to slot a3 of Calendar 0. After issuing the `Cal_Switch` command with value 0x01, the NOW pointer points to b4 of Calendar 1 as shown in Figure 7.14.

Figure 7.13 Flat Mode - Calendar Table 0

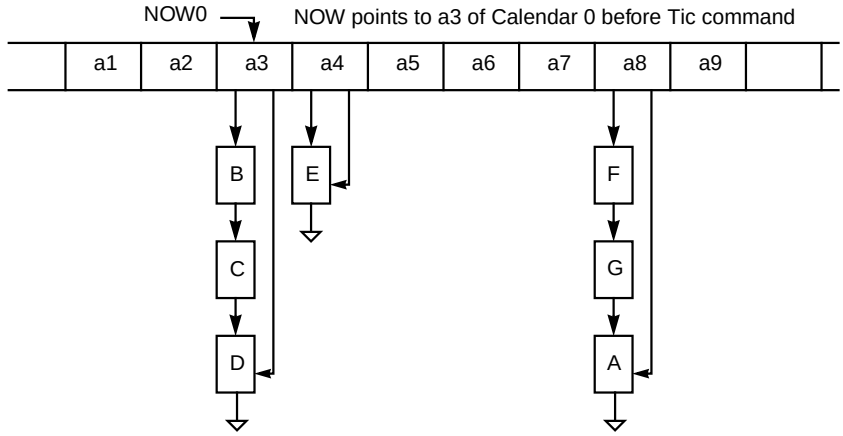
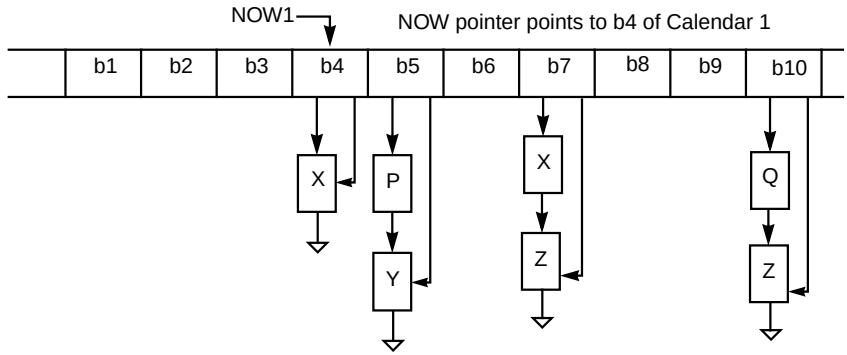


Figure 7.14 Flat Mode - Calendar Table 1



After the calendar switch, the next connection that is serviced is X from calendar 1.

The current active calendar is indicated in the Curr_Cal field (read only) of the SCD_CalSwitch register. The Cal_Base, Cal_Size, and Now pointers must be initialized correctly before switching to a calendar.

Important: The Scheduler operates either in Priority or Flat mode. Switching calendars does not change the Scheduler mode. Also, switching calendars does not change the head selection option of the priority classes.

7.5 Command Execution

The Scheduler starts command execution as soon as the APU reads or writes the appropriate hardware register. The APU may issue another command before the Scheduler completes the current one provided the new command is different from the current one. The Scheduler registers the new command and then executes it after completing the current command.

The APU stalls if it issues a command similar to the one currently being executed or is already registered for execution. To avoid such an APU stall, you must make sure that the command is not busy. The status of the SCD_Tic, SCD_Serv, SCD_Sched, and SCD_Now Scheduler commands can be verified by reading the APU_Status register (page 4-108).

The Scheduler implements a command pipe internally in case more than one command is registered so that commands get executed in the order they are received. This feature also allows the APU to issue schedule and tic commands after a service command without checking Scheduler status.

The Scheduler stores the connection(s) to be serviced in the current cell slot in its internal registers along with their SCD_BuffPres status bits. In Flat mode, there is only one connection class per calendar and in Priority mode there may be up to six connection classes for each calendar.

If the EDMA modifies the VCD_BuffPres bit (page 5-11), the Scheduler may have an outdated value. To prevent this situation, the EDMA supplies the Scheduler with the current connection number and the VCD_BuffPres bit. The Scheduler then compares the connection numbers in its internal registers for all four calendars and, if necessary, updates the corresponding SCD_BuffPres bit.

7.6 Register Descriptions

The Scheduler has the registers listed and briefly defined in Table 7.1. The APU uses an address of 0xB800.02XX to access the Scheduler registers. The Offset column in Table 7.1 specifies the value of XX for each register.

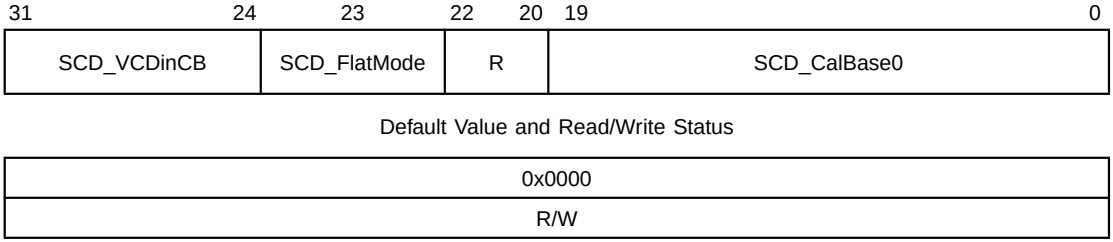
Table 7.1 Scheduler Registers

Name	Offset	Size	R/W	Description
SCD_Ctrl	0x00	32	R/W	Control Register
SCD_CalSize0	0x06	16	R/W	Size of the Calendar Table 0
SCD_Now	0x0A	16	R/W	Current Cell Slot Pointer
SCD_Serv	0x0C	32	R	Execute `service` Command
SCD_Sched	0x10	32	W	Execute `schedule` Command
SCD_Tic	0x18	32	R/W	Execute `tic` Command
SCD_CalSwitch	0x23	8	R/W	Execute `Cal_Switch` Command
SCD_CalBase1	0x28	32	R/W	Base of Calendar Table 1
SCD_CalBase2	0x2C	32	R/W	Base of Calendar Table 2
SCD_CalBase3	0x30	32	R/W	Base of Calendar Table 3
SCD_CalSize1	0x36	16	R/W	Size of the Calendar Table 1
SCD_CalSize2	0x3A	16	R/W	Size of the Calendar Table 2
SCD_CalSize3	0x3E	16	R/W	Size of the Calendar Table 3
SCD_HeadSel	0x43	8	R/W	Head Insertion Selection
SCD_Err	0x47	8	R	Error Register
SCD_Class0	0x48	32	R	Head and Tail of Priority Class 0 of Current Calendar
SCD_Class1	0x4C	32	R	Head and Tail of Priority Class 1 of Current Calendar
SCD_Class2	0x50	32	R	Head and Tail of Priority Class 2 of Current Calendar
SCD_Class3	0x54	32	R	Head and Tail of Priority Class 3 of Current Calendar
SCD_Class4	0x58	32	R	Head and Tail of Priority Class 4 of Current Calendar
SCD_Class5	0x5C	32	R	Head and Tail of Priority Class 5 of Current Calendar

7.6.1 Scheduler Control Register

The Scheduler Control register (SCD_Ctrl) provides information needed to calculate the VC Descriptor address and the Calendar Entry address. As shown in Figure 7.15, the SCD_Ctrl register includes the base address of Calendar Table 0 (SCD_CalBase0), the Scheduler operating mode (SCD_FlatMode), and the number of VCDs stored in CBM (SCD_VCDinCB).

Figure 7.15 Scheduler Control Register Format



The Scheduler operates in the Flat mode when the SCD_FlatMode bit is set and in the Priority mode when the SCD_FlatMode bit is cleared.

7.6.2 Calendar Size Register

The Scheduler's Calendar Size (SCD_CalSize0–3) registers program the sizes of the Calendar Tables in units of cell slots. The memory required to store a Calendar Table is computed as follows:

Memory (bytes) = SCD_CalSize * (2 + 2 * SCD_FlatMode)

7.6.3 SCD_Now Register

The SCD_Now register contains the current cell slot number. The Scheduler uses the current cell slot number, which is also referred to as the Calendar Index or the Now Index, to calculate the address of the Calendar Table Entry.

The Now Index resets to zero after reaching a value equal to that in the SCD_CalSize register minus 1. The APU may read the SCD_Now register to obtain the value of the Now Index.

The APU may write to the SCD_Now register to change the Calendar index. The Scheduler then reinitializes the internal head and tail pointer

registers with the connection numbers from the new cell slot in the current calendar. The Scheduler uses double buffering for the SCD_Ctrl, SCD_CalBase1-3, and SCD_CalSize0-3 registers and the values written into these registers are not used until after a write operation to the SCD_Now register.

7.6.4 SCD_Serv, SCD_Sched, and SCD_Tic Registers

The APU controls Scheduler operation by accessing the SCD_Serv, SCD_Sched, and SCD_Tic registers. When the APU reads the SCD_Serv register, the Scheduler executes the `service` command (see Section 7.2.2, “Service Command,” for Priority mode and Section 7.3.2, “Service Command,”) for Flat mode. To schedule a connection for service, the APU writes the connection number and the cell slot number into the SCD_Sched register (see Section 7.2.3, “Schedule Command,” for Priority mode and Section 7.3.3, “Schedule Command,” for Flat mode). To execute a `tic` command, the APU writes to the SCD_Tic register (see Section 7.2.4, “Tic Command”). A read from the SCD_Tic register returns the NOW pointer of the current calendar. See Section 7.4, “Calendar Switching,” for selecting different calendars.

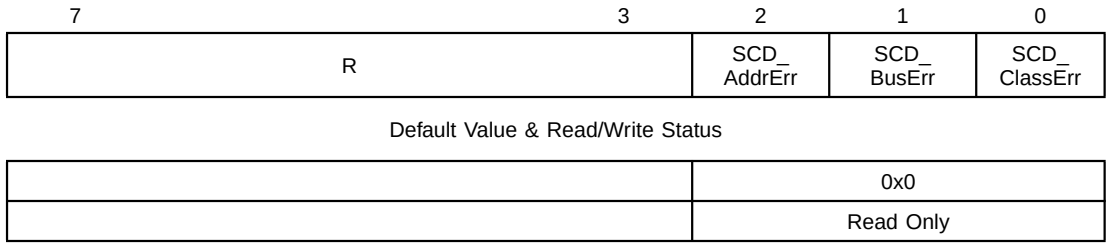
7.6.5 SCD_HeadSel Register

The SCD_HeadSel register is used to set the head insertion selection for each class in Priority mode and in Flat mode (see Section 7.2.3, “Schedule Command” and Section 7.3.3, “Schedule Command.”)

7.6.6 SCD_Err Register

The SCD_Err register holds the error conditions in the operation of the Scheduler. The error bits are set and nonvectored interrupt `IntSCD_BusErr` is issued if the interrupt is enabled. A write to the register is ignored. The bits can be cleared after they are set only by setting the `APU_Reset` bit in the APU_AddrMap register (page 4-99).

Figure 7.16 SCD_Err Register



R **Reserved** **[7:3]**
Not used in the L64364.

SCD_AddrErr **Address Error** **2**
The Scheduler sets this bit when the address it computes is not mapped in the ATMizer II+ memory map.

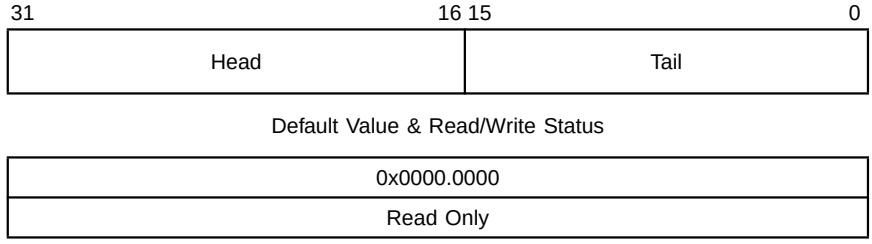
SCD_BusErr **Bus Error** **1**
The Scheduler sets this bit when it encounters a bus error when accessing the PCI or Secondary Bus.

SCD_ClassErr
Schedule Class Error **0**
This bit is set when a schedule command is issued for a connection with Class 6 or 7.

7.6.7 SCD_Class0–5 Registers

The six SCD_Class0–5 registers hold the Heads and the Tails of the corresponding priority classes of the current calendar. The Tail field of a register is valid when the Head field is nonzero. These registers are read-only and can be used to determine the internal lists held by the Scheduler. The format of the registers is shown in Figure 7.17.

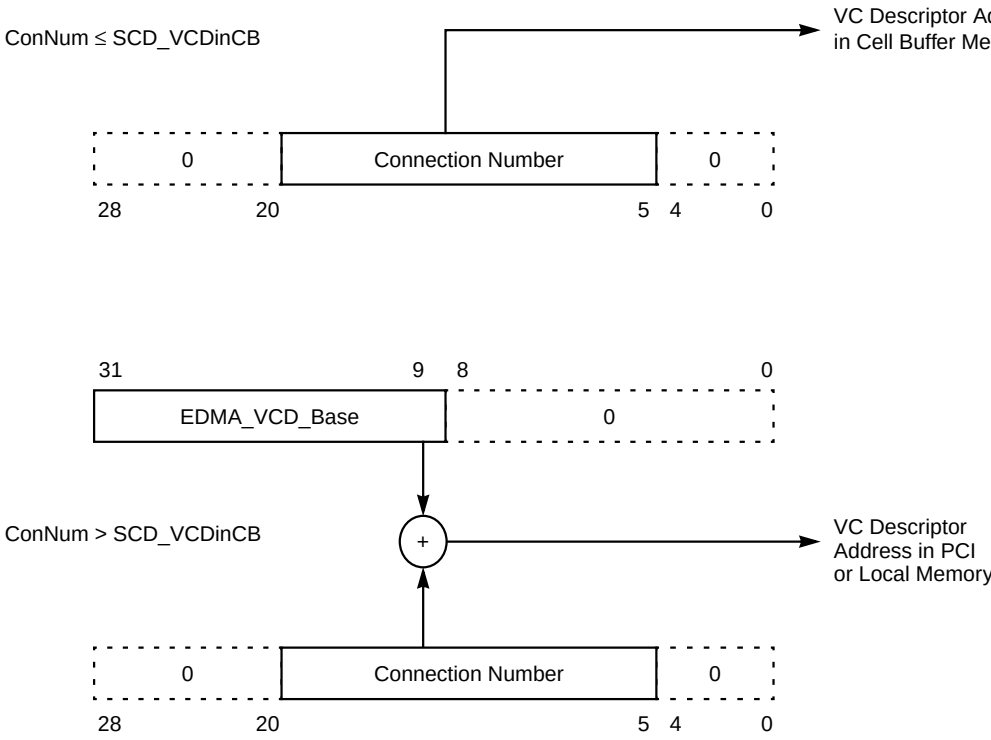
Figure 7.17 SCD_Class0–5 Registers Format



7.6.8 Calculating a VC Descriptor Address

As shown in Figure 7.18, the SCD_VCDinCB field in the SCD_Ctrl register determines the location of a VC Descriptor. Cell Buffer Memory contains the VC Descriptor if the Connection Number is less than or equal to the value in the SCD_VCDinCB field. The VC Descriptor is located in PCI or Local Memory if the connection number is greater than the value in the SCD_VCDinCB field. In the latter case, the VCD address is computed by adding the EDMA_VCD_Base to the Connection Number.

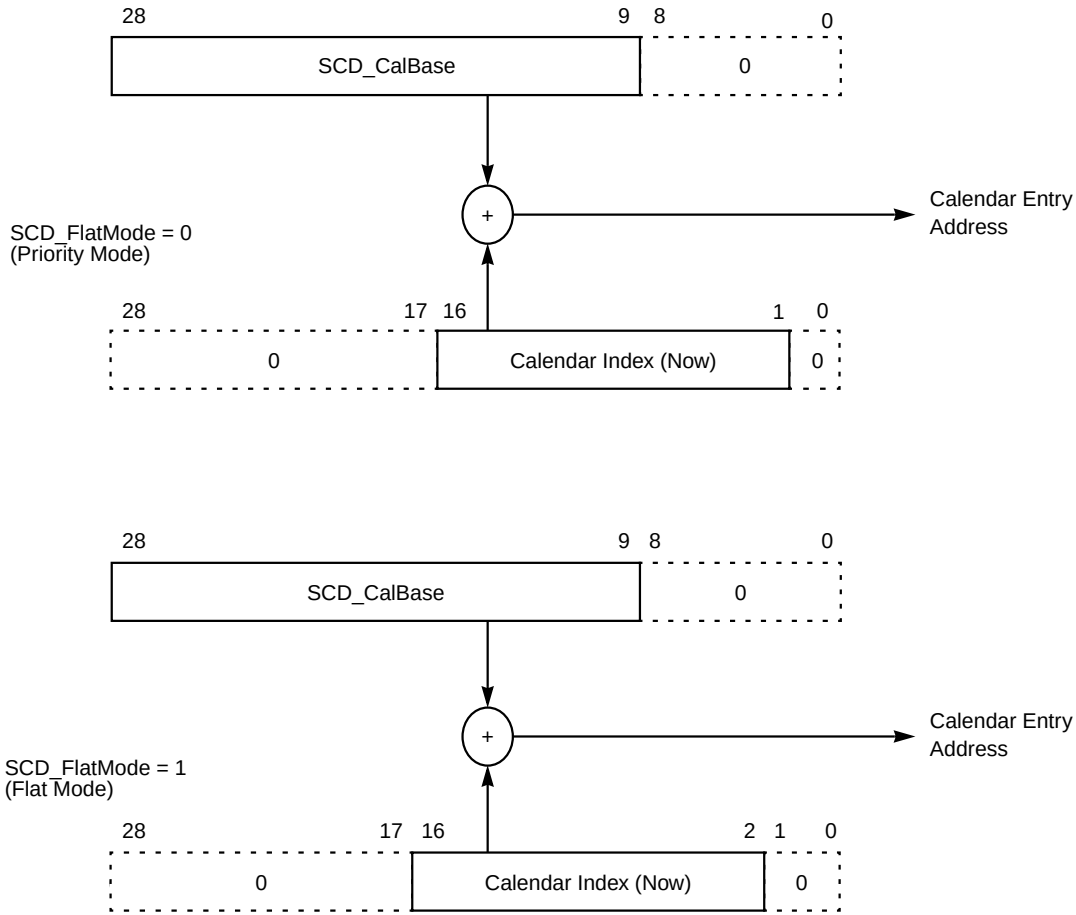
Figure 7.18 VC Descriptor Address Computations



7.6.9 Calculating a Calendar Table Address

The Scheduler computes the address of a Calendar Table Entry using the SCD_FlatMode bit, the Now index, and the SCD_CalBase as shown in Figure 7.19. If the SCD_FlatMode bit is cleared (Priority mode enabled), then the Calendar Entry Address is based on the SCD_CalBase and the Now index (which is shifted left one bit position). If the SCD_FlatMode bit is set (Flat mode enabled), then the Calendar Entry Address is based on the SCD_CalBase and the Now index (which is shifted left two bit positions). The calendars can be located only in Secondary memory or CBM.

Figure 7.19 Calendar Table Address Computations



Chapter 8
Timer Unit

This chapter describes the Timer Unit and includes the following sections:

- Section 8.1, “Introduction,” page 8-1
- Section 8.2, “Timer Clock Selection,” page 8-3
- Section 8.3, “Time-Out Events,” page 8-5

Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” should not be modified.

8.1 Introduction

The Timer Unit includes a set of hardware timers and registers that provide real-time events for the APU. There are eight general-purpose timers and a Time Stamp Counter implemented in a set of registers. The Timer Unit registers shown in Table 8.1 are accessible to the APU at physical base address 0x1800.0280.

Table 8.1 Timer Unit Registers

Name	Offset	Size	R/W	Description	Init
TM_TimeStamp	0x00	32	R/W	Time Stamp Counter	0
TM_Timer1	0x04	8	R/W	Timer Value	Yes
TM_TimerInit1	0x06	8	R/W	Timer Initialization Value	0
TM_Timer2	0x08	8	R/W	Timer Value	Yes
TM_TimerInit2	0x0A	8	R/W	Timer Initialization Value	0
TM_Timer3	0x0C	8	R/W	Timer Value	Yes
TM_TimerInit3	0x0E	8	R/W	Timer Initialization Value	0
TM_Timer4	0x10	8	R/W	Timer Value	Yes
TM_TimerInit4	0x12	8	R/W	Timer Initialization Value	0
TM_Timer5	0x14	8	R/W	Timer Value	Yes
TM_TimerInit5	0x16	8	R/W	Timer Initialization Value	0
TM_Timer6	0x18	8	R/W	Timer Value	Yes
TM_TimerInit6	0x1A	8	R/W	Timer Initialization Value	0
TM_Timer7	0x1C	8	R/W	Timer Value	Yes
TM_TimerInit7	0x1E	8	R/W	Timer Initialization Value	0
TM_Enable	0x20	6	R/W	Time-Out Enable	Yes
TM_Clear	0x24	6	W	Time-Out Clear	–
TM_ClockSel	0x28	32	R/W	Timer Clock Selection	Yes
TM_ClockSel2	0x2C	8	R/W	Timer Clock Selection 2	Yes
TM_Timer8	0x30	8	R/W	Timer Value	0
TM_TimerInit8	0x32	8	R/W	Timer Initialization Value	Yes

At initialization time, the Time Stamp Counter (TM_TimeStamp register) has a count of zero. The counter increments once for each timing event until it reaches a maximum count of 0xFFFF.FFFF and then wraps around to zero. Each general-purpose timer register (TM_Timer 1 through 8) starts with a value programmed in it and then decrements

once for each timing event until it reaches a count of zero (times out). Then the value from the corresponding TimerInit register is loaded into the timer register.

The clock selection registers (TM_ClockSel and TM_ClockSel2) specify the timing event for the Time Stamp Counter and each general-purpose timer register. A timing event can be an external clock, the internal system clock (see Section 11.1, “System Clock Options”), or a time-out event of another general-purpose timer.

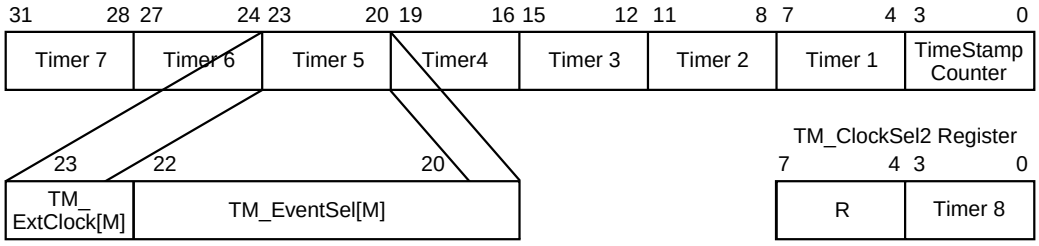
A time-out event occurs when a general-purpose timer reaches a count of zero. Time-out events for Timers 1 through 3 and 8 are reported to the APU using the APU_Status register. The eight general-purpose timers may be cascaded to achieve higher counts.

8.2 Timer Clock Selection

All timers have input clocks that are selectable using the TM_ClockSel (Timer Clock Selection) register shown in Figure 8.1. The TM_ClockSel register is divided into eight 4-bit fields. Bits [3:0] select the clock and time-out event for the Time Stamp Counter. Bits [31:4] select the clocks and events for general-purpose timers 1 through 7. Bits [3:0] of TM_ClockSel2 register are used to select the clock and time-out event for timer 8. Note that the time-out event of timer 8 cannot be used as a clocking event for timers 1–7 and the Time Stamp Counter.

Both registers default to all zeros and both are read/write registers. The TM_ClockSel register is located at address 0xB800.02A8 and the TM_ClockSel2 register is located at address 0xB800.02AC.

Figure 8.1 Timer Clock Selection Registers Format



8.2.1 TM_ClockSel Register

TM_ExtClock[M]

Timer External Clock Select **[31,27,...,3]**
When set, an external clock is used as the clock for timer M or the Time Stamp Counter; when cleared, the internal system clock (see Section 11.1, “System Clock Options”) is used. The L64364 has an external clock input for this purpose.

TM_EventSel[M]

Timer Event Select **[30:28],[26:24],...,[2:0]**
The values in these 3-bit fields determine when the selected clocks decrement a general-purpose timer or increment the Time Stamp Counter. When set to 0b000, every selected clock pulse is used. When set to 0b001 through 0b111, the time-out event for the corresponding timer gates a selected clock pulse to update the timer/counter. For example, if Timer 2 is set to 1, Timer 2 is decremented once for every time-out of Timer 1.

Notes: The time-out event of Timer 8 cannot be used as a clocking event for Timers 1–7 or the Time Stamp Counter.

Since an output of any general-purpose timer (except Timer 8) may be used as an input of another timer, it is possible to create a loop that will result in all of the timers in the loop being stalled. The Timer Unit does not check for this condition.

8.2.2 TM_ClockSel2 Register

R **Reserved** **[8:5]**
Not used in the L64364.

Timer 8 **Timer 8 Clock and Event Select** **[4:0]**
When bit 3 is set, an external clock is used to toggle Timer 8. When bit 3 is cleared, the system clock is used. Bits [2:0] determine when the selected clock decrements Timer 8. When set to 0b000, every selected clock pulse is used. When set to 0b001 through 0b111, the time-out event for the corresponding timer gates a selected clock pulse to update Timer 8.

8.3 Time-Out Events

The APU_Status register (page 4-108) records unmasked time-out events for Timers 1 through 3 and Timer 8. See Section 4.8.5, “Status Checking.”

Since general-purpose timers may be cascaded, it is often unnecessary to report time-out events of all timers. Time-out events for only timers 1 to 3 and timer 8 are provided. The TM_Enable register may be used to mask some of them. Bits [3:1] enable Timers 3 to 1 time-outs and bit 0 enables Timer 8. When a bit is cleared, the time-out event is not registered in the APU_Status register; when the bit is set, the event is registered and can generate an interrupt.

The APU may read the APU_Status register to check which timer has timed-out. The read operation does not change the contents of the register. The APU cannot directly set or clear a bit in the APU_Status register. To clear time-out event bits [3:0] of the APU_Status register, the APU must access the TM_Clear register. Writing a one to a bit position in the TM_Clear register clears the corresponding time-out event bit in the APU_Status register. Writing zero leaves the bit unchanged.

The APU can access each general-purpose timer register, the TM_TimeStamp register, and the associated Timer Initialization registers (see Section 8.1, “Introduction”). The EDMA copies the current value of the TM_TimeStamp register to a VC Descriptor when it executes a `cell` command for the receive direction.

Time-out events for General-Purpose Timers 4 through 7 as well as the TimeStamp Counter are not recorded in the APU_Status register. These timers can only be used as part of a wider, cascaded timer.

Chapter 9
PCI Interface

This chapter describes the L64364 PCI Interface and contains the following sections:

- Section 9.1, “PCI Interface Overview,” page 9-1
- Section 9.2, “PCI Configuration Space Registers,” page 9-4
- Section 9.3, “Primary Port Registers,” page 9-21
- Section 9.4, “PCI Slave Transactions,” page 9-29
- Section 9.5, “PCI Master Transactions,” page 9-36
- Section 9.6, “Balancing Bus Usage,” page 9-44

This chapter assumes that the reader is familiar with the contents of the *PCI Local Bus Specification* and does not repeat information contained in that specification.

Important: Register bits and fields labeled “Reserved” are don’t cares. Descriptor bits and fields labeled “Reserved” should not be modified.

9.1 PCI Interface Overview

The PCI Interface, illustrated in Figure 9.1, includes the following major functional units:

- Four FIFOs for PCI Bus read and write transactions:
  - PCI Bus Master Read FIFO
  - PCI Bus Master Write FIFO
  - PCI Bus Slave Read FIFO
  - PCI Bus Slave Write FIFO

- 17 PCI configuration registers
- Two PCI control registers
- MailBox FIFO

The PCI Interface has the following features:

- Conforms to *PCI Local Bus Specification, Revision 2.1.*
- Handles 32-bit master and slave transactions at 33 MHz.
- Performs read/write master transactions to/from the APU and EDMA.
- Performs read/write slave transactions to/from the Mailbox FIFO, Cell Buffer Memory (CBM), and local memory (that is, Secondary Bus Memory).
- Includes independent master and slave state machines, and four FIFOs which support interleaving concurrent master and slave PCI Bus transactions.
- Supports a PCI system architecture based on a producer-consumer model for optimal PCI Bus performance:
  - PCI host (producer) writes ATM transmit payload data to the L64364 local memory for segmentation (slave write transaction).
  - The L64364 writes incoming data (received from the Utopia Bus) to the PCI host (consumer) using master write transactions.
- Alternatively, the L64364 supports master read and write transfers of transmit and receive PDUs to/from PCI memory.
- Supports segmentation and reassembly in either PCI memory or the L64364 local memory.
- The PCI Configuration register may be accessed by both an external PCI configuration host and the APU. See Section 9.2.20 and Section 9.2.21 for more details. The L64364 can also drive PCI configuration cycles.

Figure 9.1 PCI Interface Block Diagram

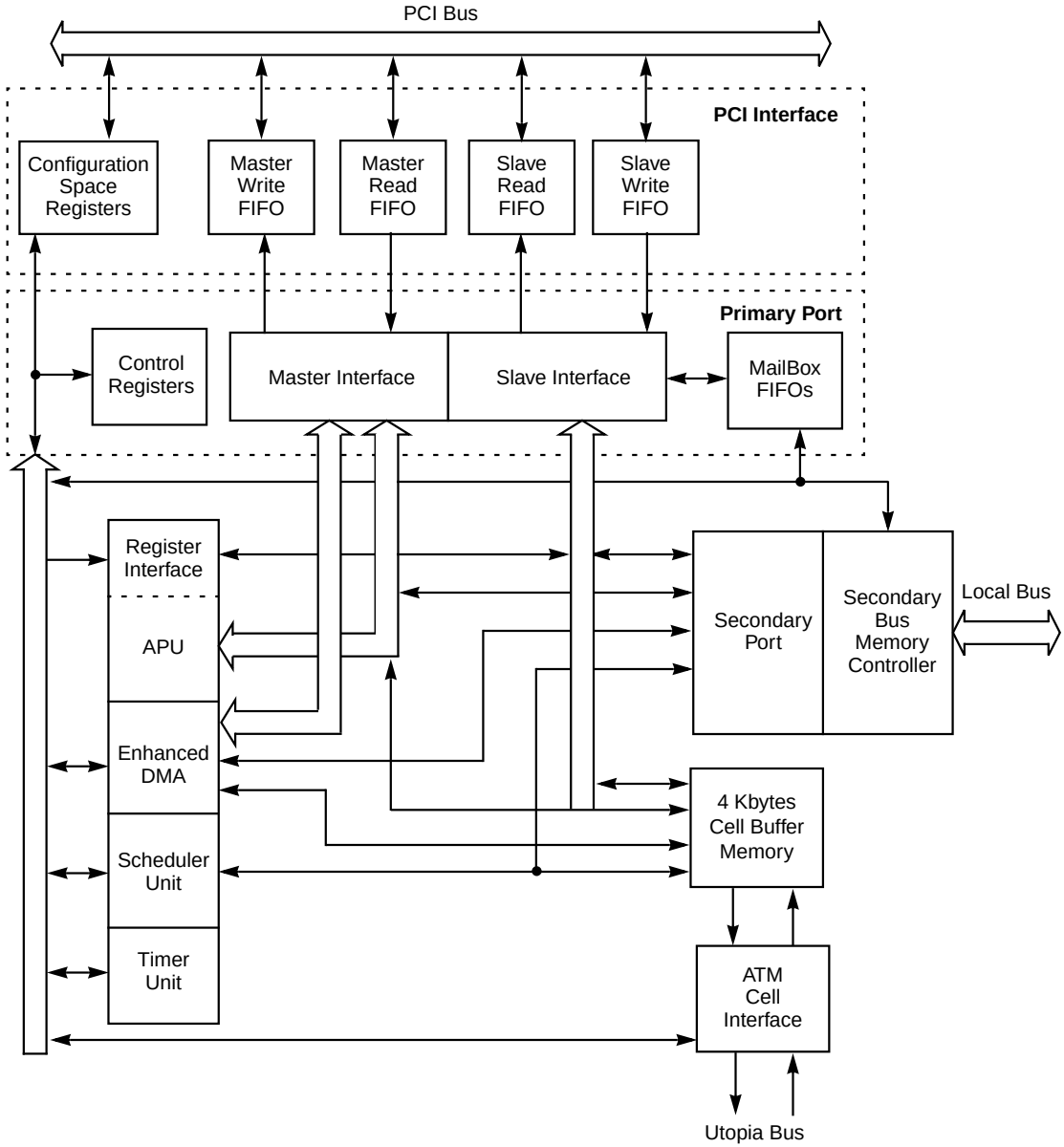


Table 9.1 summarizes the functions and sizes of the FIFOs contained in the PCI Interface.

Table 9.1 PCI FIFO's

FIFO	Function	Size
Master Read	APU or EDMA read of PCI memory	32 x 32
Master Write	APU or EDMA write to PCI memory	32 x 32
Slave Read	PCI read of CBM, Mailbox, or local memory[1]	8 x 32
Slave Write	PCI write to CBM, Mailbox, or local memory	32 x 32

1. Only single word reads are allowed to the Mailbox and L64364 registers. The Slave Read FIFO is provided for local memory and CBM access.

9.2 PCI Configuration Space Registers

PCI Configuration Space registers configure and control the PCI Interface. These registers are accessible through the PCI configuration space as defined in Figure 9.2 and by the APU at addresses 0xB800.0900 through 0xB800.0944 in the APU hardware register space.

Notes: The APU accesses the registers in big endian format.

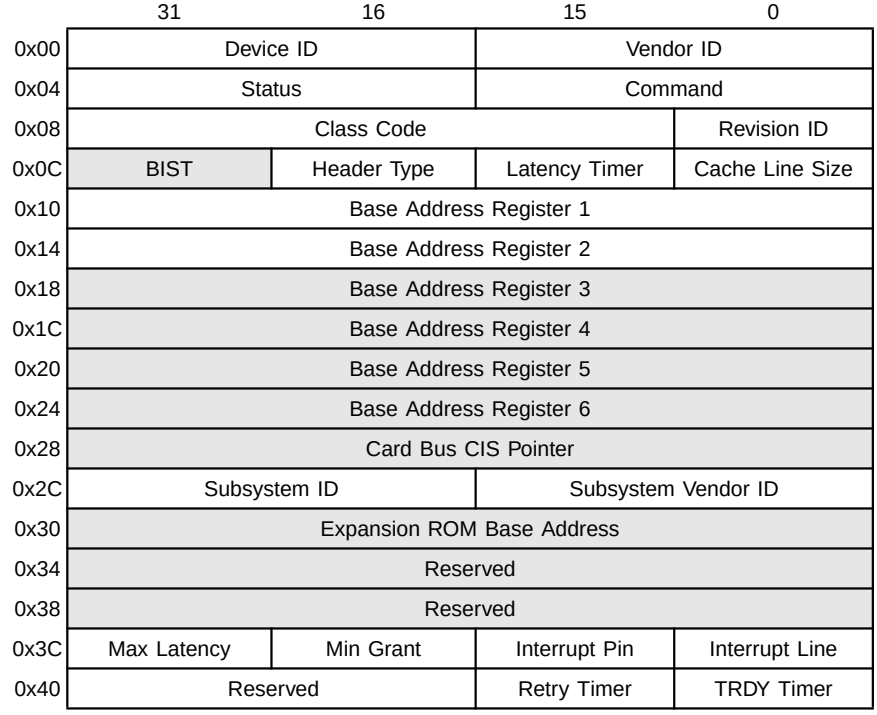
It is your responsibility to initialize the `SB_64M` bit in the SP_Ctrl register (affects the PCI Base Address register 2) and the PCI Subsystem_ID and Subsystem_Vendor_ ID registers to appropriate values before allowing external PCI hosts access to the L64364 PCI Configuration Space. See Section 9.2.21, "Configuration Master Operation" and Section 9.3.2, "PP_Ctrl Register."

In addition to the PCI Configuration registers, three other registers control PCI Interface operation. Maximum burst size on the PCI Bus and Secondary Bus is set by the APU in the PP_Ctrl register. PCI burst size is limited to the maximum allowed by the PP_Ctrl register or PCI latency timer. The amount of data to prefetch for slave read requests is set by the APU in the PP_SlavePFtch register. The XPP_Ctrl register is accessible in the PCI memory space and allows a PCI host to control APU boot and PCI interrupts. These registers are defined in Section 9.3, "Primary Port Registers."

The L64364 supports Type 0 Configuration Space access. Figure 9.2 provides a map of the PCI Configuration Space registers. The shaded registers are not used by the L64364. If a write transfer to an unused register occurs, the L64364 handles it as a normal operation but ignores the data. If a read transfer from an unused register occurs, the L64364 handles it as a normal operation but all the data will be zero.

Note: The little endian format is used in the PCI Configuration Space registers (that is, byte 0 is the LSB).

Figure 9.2 PCI Configuration Space Registers

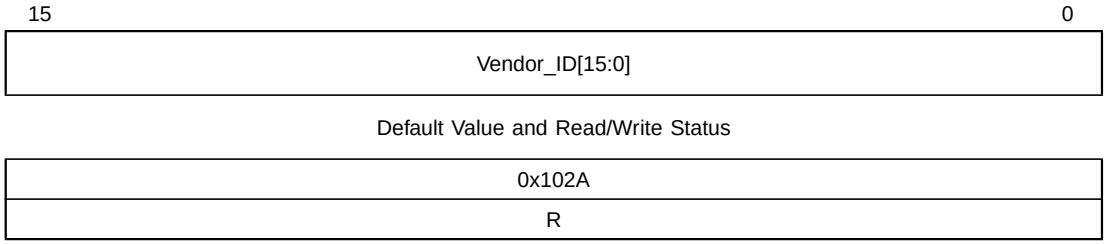


The following sections describe the Configuration Space registers. Registers reset to the default values identified when PCI_RSTn is asserted. PCI control bits not used by the L64364 are shaded.

9.2.1 Vendor ID Register

This register, illustrated in Figure 9.3, contains the LSI Logic vendor ID and is located at Configuration Space address offset 0x00.

Figure 9.3 Vendor ID Register



Vendor_ID[15:0]

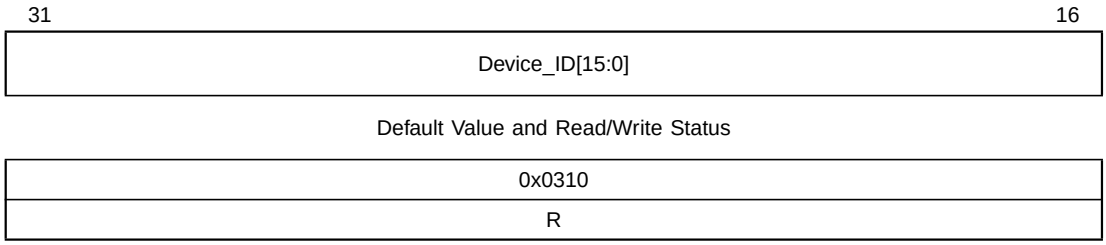
PCI Vendor Identification Number [15:0]

The LSI Logic Vendor ID as specified by the PCI Special Interest Group is 0x102A.

9.2.2 Device ID Register

This register, illustrated in Figure 9.4, contains the device ID for the L64364 and is located at Configuration Space address offset 0x02.

Figure 9.4 Device ID Register



Device_ID[15:0]

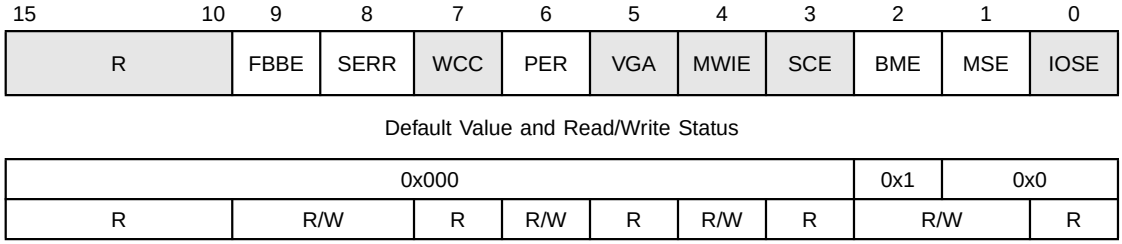
PCI Device Identification [31:16]

The L64364 has a Device ID of 0x0310.

9.2.3 Command Register

This register, illustrated in Figure 9.5, contains commands that control the PCI Interface. It is located at Configuration Space address offset 0x04.

Figure 9.5 Command Register



R **Reserved** **[15:10]**
Not used in the L64364.

FBBE **Fast Back-to-Back Enable** **9**
When set, the L64364 is enabled to drive fast back-to-back transactions.

SERR **System Error Enable** **8**
When set, SERR enables assertion of PCI_SERRn when an address parity error is detected. PER (bit 6) must also be set to enable parity detection.

WCC **Wait Cycle Control** **7**
The L64364 does not support address/data stepping. This bit must remain cleared.

PER **Parity Error Response** **6**
When set, PER enables detection of PCI address and data parity error responses. Per the *PCI Local Bus Specification*, all agents are required to generate valid parity.

When the L64364 detects an address parity error, it claims the cycle and terminates the transfer with target abort. If SERR (bit 8) is set, the L64364 also asserts the PCI_SERRn signal when an address parity error is detected.

When the L64364 detects a data parity error, it completes the data transfer and asserts PCI_PERRn and the IntPCIErr interrupt if the PER bit is set.

VGA **VGA Palette Snoop** **5**
Not used in the L64364. This bit should remain cleared.

MWIE **Memory Write and Invalidate Enable** **4**
Memory write and invalidate is not supported in the L64364. This bit must remain cleared.

SCE **Special Cycle Enable** **3**
Special cycles are not supported in the L64364. This bit must remain cleared.

BME **Bus Master Enable** **2**
When set, the L64364 can act as PCI Bus master. When cleared, the L64364 will not request the PCI Bus. The L64364 will, however, respond to slave transactions.

This bit is set at initialization.

MSE **Memory Space Enable** **1**
MSE enables the L64364 to respond to slave transactions targeting the memory space selected by Base Address Register 1 or Base Address Register 2.

IOSE **I/O Space Enable** **0**
The L64364 does not use I/O Space. This bit must remain cleared.

9.2.4 Status Register

This register, illustrated in Figure 9.6, indicates the status of the PCI Interface operations and data. It is located at Configuration Space address offset 0x06.

Figure 9.6 Status Register

31	30	29	28	27	26 25	24	23	22	21	20 16
DPE	SSE	RMA	RTA	STA	DEVSEL[1:0]	DPED	FBBC	UDFS	66MC	R

Default Value and Read/Write Status

0x00	0x1	0x0	0x1	0x00
R/W	R	R/W	R	

Note: To clear a R/W bit in the Status register, write a 1 to that bit position. Writing a 0 to a R/W bit in the Status register leaves the bit unchanged.

DPE **Detected Parity Error** **31**
DPE is set when address or data parity errors are detected, independent of the parity enable bits in the Command register.

SSE **Signaled System Error** **30**
SSE is set when the L64364 asserts the PCI_SERRn signal.

RMA **Received Master Abort** **29**
RMA is set when the L64364 terminates a transfer with Master Abort. This condition occurs if PCI_DEVSELn is not asserted within six PCI clocks from the time the L64364 asserts PCI_FRAMEn during Master Read or Master Write transactions.

RTA **Received Target Abort** **28**
The L64364 sets RTA when a Master Read or Master Write transaction is terminated by a Target Abort.

STA **Signalled Target Abort** **27**
The L64364 signals Target Abort by setting this bit only when it detects an address parity error on an access to its memory spaces as defined by the Base Address 1 and Base Address 2 registers.

DEVSEL[1:0] **DEVSEL Timing** **[26:25]**
DEVSEL[1:0] specifies the maximum number of PCI cycles required for the L64364 to assert PCI_DEVSELn in response to PCI_FRAMEn during Slave Read/Write transactions. DEVSEL[1:0] is hardwired to 0b01 for medium timing.

DPED **Data Parity Error Detected** **24**
DPED is set when the L64364 detects PCI_PERRn asserted during Master Read or Master Write transactions and the PER bit (bit 6 in the Command register) is set.

FBBC **Fast Back-to-Back Capable** **23**
The L64364 is Fast Back-to-Back capable. This bit will always be 1.

UDFS **User-Defined Features Supported** **22**
User-Defined Features are not supported in the L64364. This bit will always be 0.

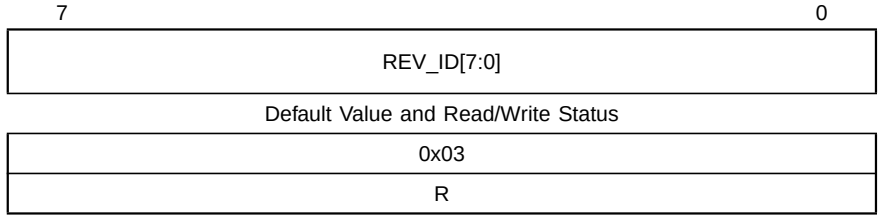
66MC **66 MHz Capable** **21**
The L64364 does not support the 66 MHz PCI Bus. This bit will always be 0.

R **Reserved** **[20:16]**
Not used in the L64364.

9.2.5 Revision ID Register

This register, illustrated in Figure 9.7, contains the current revision ID for the L64364 and is located at Configuration Space address offset 0x08.

Figure 9.7 Revision ID Register

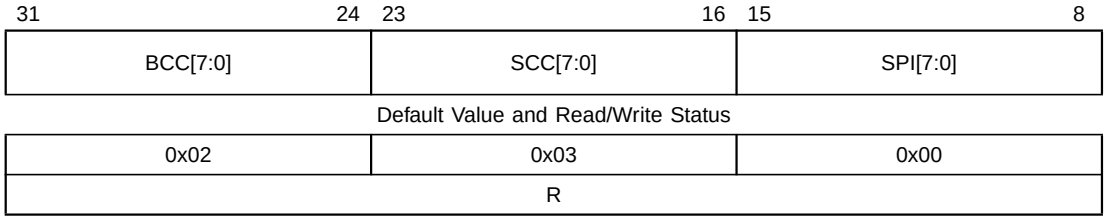


REV_ID[7:0] **Revision ID** **[7:0]**
The current L64364 revision ID, 0x03, is loaded into this register at boot up.

9.2.6 Class Code Register

This register, illustrated in Figure 9.8, contains the ATM controller class codes for the L64364. It is located at Configuration Space address offset 0x09.

Figure 9.8 Class Code Register



BCC[7:0] **Base Class Code** **[31:24]**
The L64364 is classified as a network controller. BCC[7:0] is hardwired to 0x02.

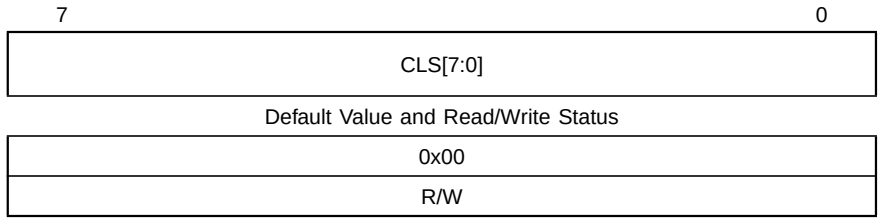
SCC[7:0] **Subclass Code** **[23:16]**
The subclass code for the L64364 is 0x03, ATM controller.

SPI[7:0] **Specific Programming Interface** **[15:8]**
There are no specific programming interfaces defined for ATM controllers. SPI[7:0] is hardwired to 0x00.

9.2.7 Cache Line Size Register

This register, illustrated in Figure 9.9, specifies the size of a cache line and is located at Configuration Space address offset 0x0C.

Figure 9.9 Cache Line Size Register

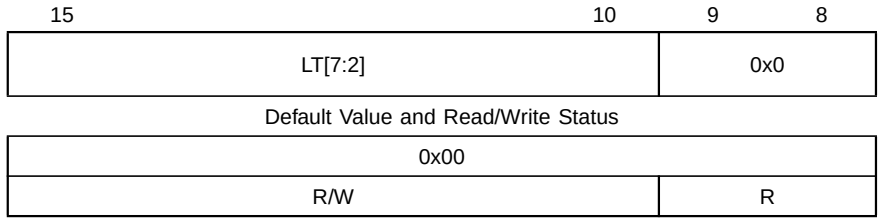


CLS[7:0] **Cache Line Size** **[7:0]**
CLS[7:0] specifies the number of 32-bit words in the system cache line. The L64364 uses CLS[7:0] to determine whether to issue Memory Read or Memory Read Line commands on Master Read transactions.

9.2.8 Latency Timer Register

This register, illustrated in Figure 9.10, specifies how long the L64364 retains bus ownership after a Master Read or Write transaction. It is located at Configuration Space address offset 0x0D.

Figure 9.10 Latency Timer Register

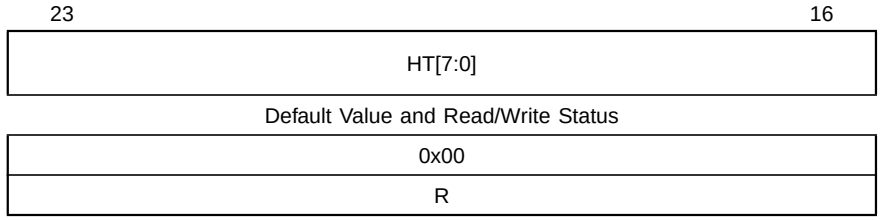


LT[7:0] **Latency Timer Value** **[15:8]**
LT[7:0] specifies the maximum number of PCI clocks that the L64364 may retain bus ownership after PCI_GNTn is deasserted on Master Read and Write transfers. The two LSBs are hardwired to 0 to provide a granularity of 4 clocks.

9.2.9 Header Type Register

This register, illustrated in Figure 9.11, specifies the Header type and is located at Configuration Space address offset 0x0E.

Figure 9.11 Header Type Register

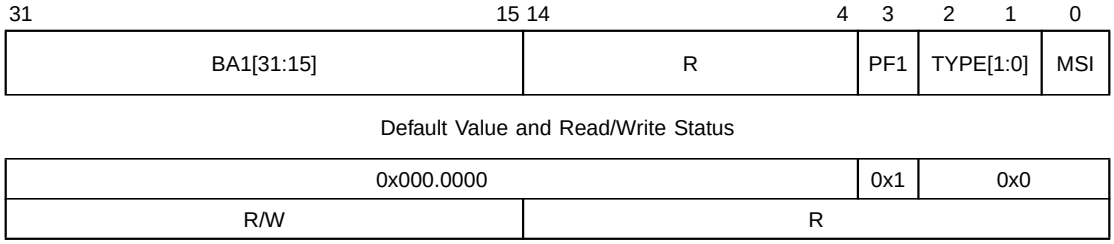


HT[7:0] **Header Type** **[23:16]**
HT[7:0] is hardwired to 0x00, indicating the L64364 supports Type 0 Configuration Space accesses.

9.2.10 Base Address Register 1

This register, illustrated in Figure 9.12, specifies parameters for Cell Buffer Memory, Mailbox, and L64364 register addressing. It is located at Configuration Space address offset 0x10.

Figure 9.12 Base Address Register 1



BA1[31:15] **Base Address 1** **[31:15]**
BA1[31:15] specifies the PCI Base Address for the ATMizer's Cell Buffer Memory, MailBox, XPP_Cntl register, and APU register space. The L64364 uses 32 Kbytes of PCI memory space for access to these items.

R **Reserved** **[14:4]**
Not used in the L64364.

PF1 **Prefetchable** **3**
The PF1 bit is hardwired to 1 indicating local memory is prefetchable.

Note: Read accesses to the Mailbox, XPP_Ctrl register, and APU registers is limited to one word at a time. Read accesses to CBM prefetch the number of words specified in the PP_SlavePFtch register. See Section 9.3.3, "Primary Port Slave Prefetch Register."

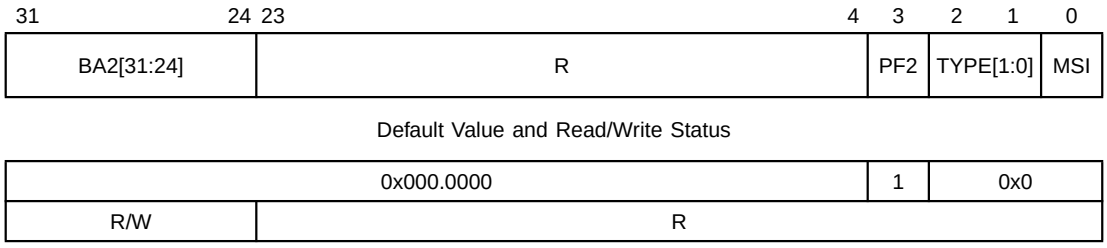
TYPE[1:0] **Base Address Type** **[2:1]**
TYPE[1:0] is hardwired to 0b00, indicating that Base Address register 1 can be located anywhere in the 32-bit PCI memory space.

MSI **Memory Space Identifier** **0**
MSI is hardwired to 0, indicating that Base Address 1 is in Memory Space rather than I/O Space.

9.2.11 Base Address Register 2

This register, illustrated in Figure 9.13, specifies parameters for local memory addressing. It is located at Configuration Space address offset 0x14.

Figure 9.13 Base Address Register 2



BA2[31:24] **Base Address 2** **[31:24]**

BA2[31:24] specifies the PCI Base Address for local memory connected to the ATMizer's Secondary Bus. From 16 Mbytes to 64 Mbytes of local memory is accessible by the PCI Bus under control of the SB_64M bit in the SP_Ctrl register (page 10-5).

When SB_64M is cleared, a 16 Mbyte local memory map is selected and BA2[31:24] define the PCI Base Address. If SB_64M is set, a 64 Mbyte local memory map is selected and BA2[31:26] define the PCI Base Address. SB_64M should be programmed before Base Address Register 2 is accessed.

R **Reserved** **[23:4]**

Not used in the L64364.

PF2 **Prefetchable** **3**

The L64364 allows multiple-word bursts from local memory. The PF2 bit is hardwired to 1, enabling local memory to be prefetchable.

Note: Read accesses to the Secondary Bus prefetch the number of words specified in the PP_SlavePFtch register. See Section 9.3.3, "Primary Port Slave Prefetch Register."

TYPE[1:0] **Base Address Type** **[2:1]**

TYPE[1:0] is hardwired to 0x0, indicating that Base Address Register 2 can be located anywhere in the 32-bit PCI memory space.

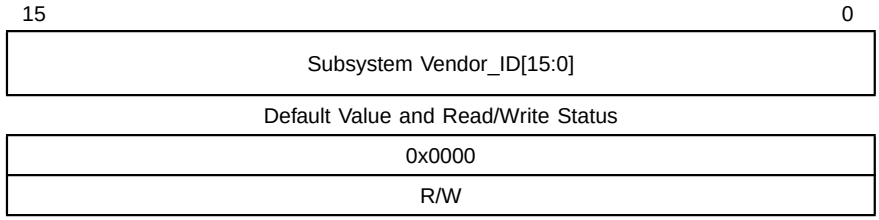
MSI **Memory Space Identifier** **0**

MSI is hardwired to 0, indicating that Base Address Register 2 is in Memory Space rather than I/O Space.

9.2.12 Subsystem Vendor ID Register

This register, illustrated in Figure 9.14, contains the Subsystem Vendor ID number and is located at Configuration Space address offset 0x2C.

Figure 9.14 Subsystem Vendor ID Register



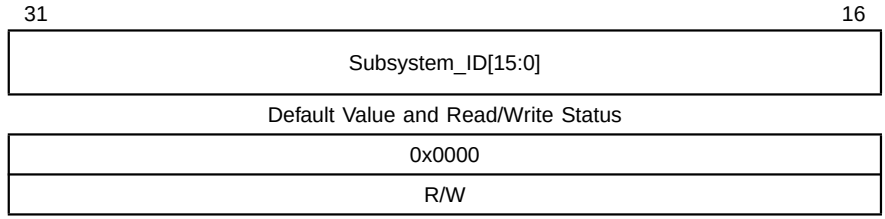
Subsystem Vendor_ID[15:0] [15:0]

The Subsystem Vendor_ID register is read only from the PCI interface. It resets to a default value of 0, and it is up to you to program the correct information from the APU. The Subsystem Vendor_ID can be read from or written to by the APU through the APU PCI_Subsystem_ID register (0xB800.092E).

9.2.13 Subsystem ID Register

This register, illustrated in Figure 9.15, contains the Subsystem ID number and is located at Configuration Space address offset 0x2E.

Figure 9.15 Subsystem ID Register



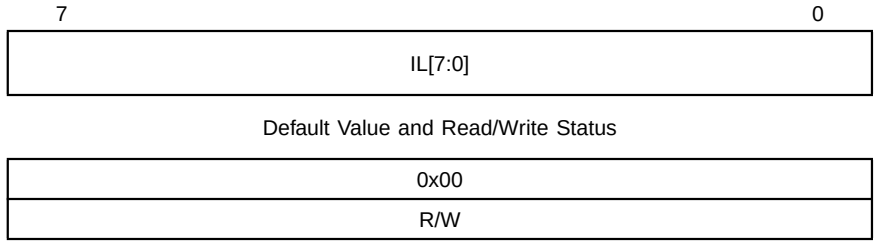
Subsytem_ID[15:0] [31:16]

The Subsytem_ID register is read only from the PCI interface. It resets to a default value of 0, and it is up to you to program the correct information from the APU. The Subsystem ID can be read from and written to by the APU through the APU PCI_Subsystem_ID register (0xB800.092C).

9.2.14 Interrupt Line Register

This register, illustrated in Figure 9.16, contains the interrupt line number, IL, and is located at Configuration Space address offset 0x3C.

Figure 9.16 Interrupt Line Register

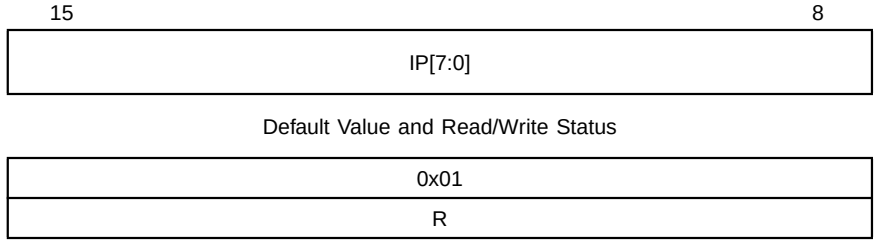


IL[7:0] **Interrupt Line** **[7:0]**
The IL register is provided for PCI system initialization. The value of IL[7:0] has no effect on L64364 operation.

9.2.15 Interrupt Pin Register

This register, illustrated in Figure 9.17, specifies the interrupt pin assignment and is located at Configuration Space address offset 0x3D.

Figure 9.17 Interrupt Pin Register

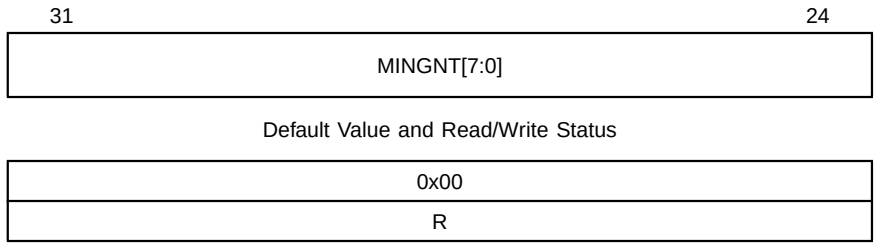


IP[7:0] **Interrupt Pin** **[15:8]**
Since the L64364 is a single function device, IP[7:0] is hardwired to 0x01 which indicates that PCI_INTn corresponds to the INTA# signal defined in the *PCI Local Bus Specification*.

9.2.16 Minimum Grant Register

This register, illustrated in Figure 9.18, contains a value for minimum grant duration. It is located at Configuration Space address 0x3E.

Figure 9.18 Minimum Grant Register

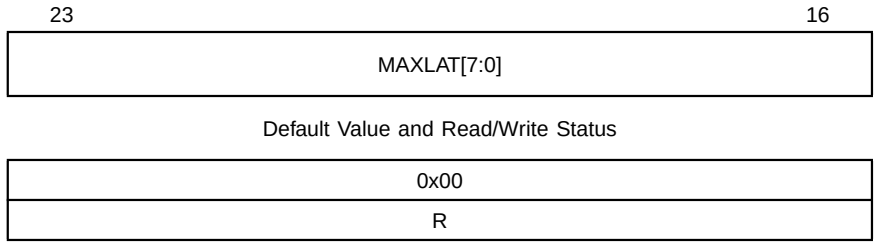


MINGNT[7:0] Minimum Grant [31:24]
No specific requirements or recommendations are made for minimum PCI grant duration. `MINGNT[7:0]` is hardwired to 0x00.

9.2.17 Maximum Latency Register

This register, illustrated in Figure 9.19, contains a hardwired latency value and is located at Configuration Space address offset 0x3F.

Figure 9.19 Maximum Latency Register

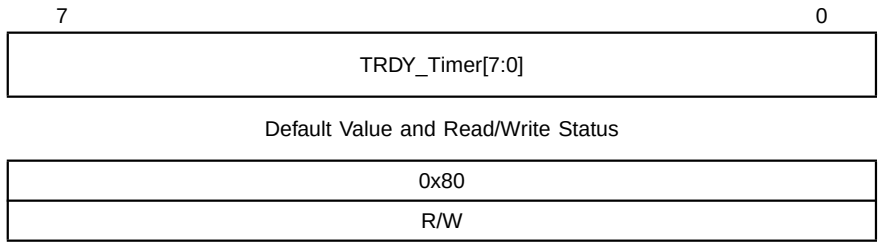


MAXLAT[7:0] Maximum Latency [23:16]
No specific requirements or recommendations are made for maximum PCI Bus latency. `MAXLAT[7:0]` is hardwired to 0x00.

9.2.18 TRDY_Timer Register

The TRDY_Timer register, illustrated in Figure 9.20, is used to recover from a target that asserts DEVSELn but does not assert TRDYn or STOPn. This register contains the maximum number of clock cycles that a target has to assert TRDYn or STOPn after FRAMEn is asserted. If the target does not respond within the specified number of clocks, the L64364 removes FRAMEn and IRDYn to get off the PCI Bus. It also signals an internal bus error to the requesting master and, if enabled, asserts IntPCIErr (nonvectored interrupt 5). The register is located at Configuration Space address offset 0x40. A value of zero disables the TRDY Timer.

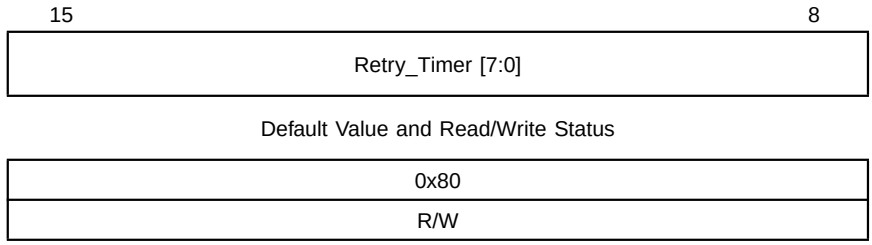
Figure 9.20 TRDY_Timer Register



9.2.19 Retry_Timer Register

The Retry_Timer register, illustrated in Figure 9.21, is used to recover from a target that continually responds to a request with retry (STOPn asserted). This register contains the maximum number of retries that the L64364 allows on any master request. When the retry limit is reached, an internal bus error is signaled to the requesting master and, if enabled, IntPCIErr (nonvectored interrupt 5) is asserted. The register is located at Configuration Space address 0x41. A value of 0 disables the Retry Timer.

Figure 9.21 Retry_Timer Register



9.2.20 Configuration Target Operation

It is your responsibility to initialize the SB_64M bit in the SP_Ctrl register (affects the PCI Base Address register 2) and the PCI Subsystem_ID and Subsystem_Vendor_ID configuration registers to appropriate values before allowing external PCI hosts to access the L64364 PCI Configuration Space. These registers are writeable only through the APU hardware register space (0xB800.09xx). PCI target accesses to the L64364 can be held off by setting the PCI_Hold bit in the PP_Ctrl register. When this bit is set, the L64364 responds with retries to all target requests. This prevents an external host from accessing the L64364 PCI Configuration Space before it is properly initialized.

Figure 9.22 and Figure 9.23 illustrate the timing of PCI configuration cycles directed to the L64364. To initiate the configuration cycle, an external host asserts PCI_IDSEL, PCI_FRAMEn, a configuration read or write command (PCI_CBEn[3:0] = 0xA or 0xB), and an appropriate configuration register address. PCI_IDSEL need only be valid during the PCI address phase. In response, the L64364 asserts PCI_DEVSELn two clock edges later (medium DEVSEL timing).

The external host then asserts PCI_IRDYn after it drives valid data on PCI_AD[31:0] and valid byte enables on PCI_CBEn[3:0] for a configuration write command, or when it is ready to receive read data from PCI_AD[31:0] on a configuration read command. The L64364 asserts PCI_TRDYn when it has accepted configuration write data from PCI_AD[31:0] or when it has placed configuration read data on PCI_AD[31:0].

The relationship between PCI_FRAMEn and PCI_TRDYn in the two figures is typical and not exact.

Figure 9.22 Configuration Space Read

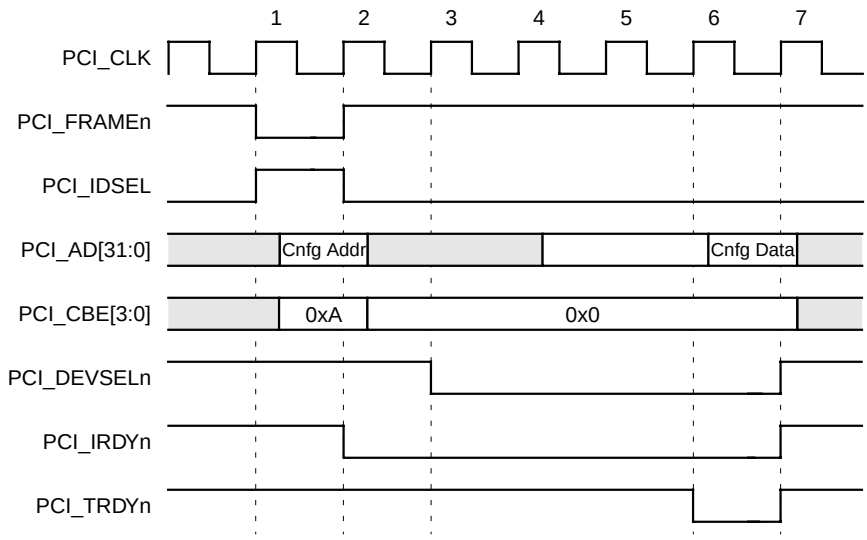
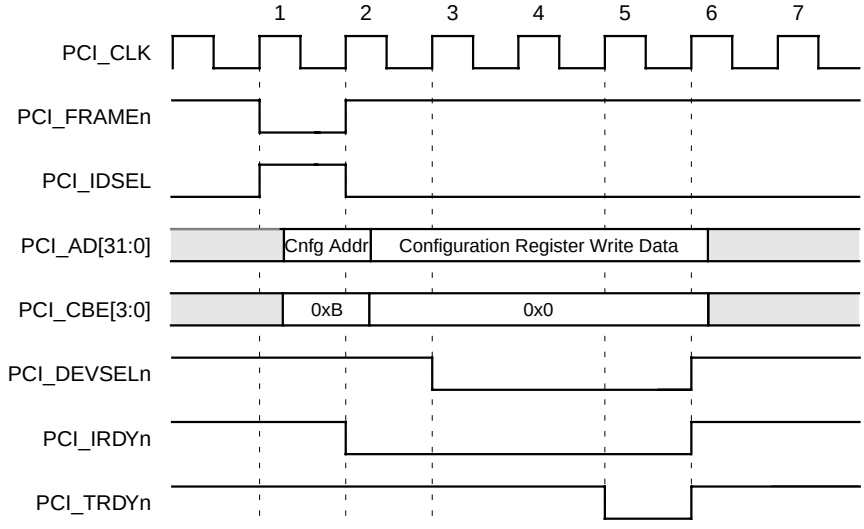


Figure 9.23 Configuration Space Write



9.2.21 Configuration Master Operation

The APU accesses the L64364 PCI configuration registers using APU hardware register space 0xB80009xx, and drives external PCI Configuration cycles when the `PCI_Cfg` bit of the PP_Ctrl register

converts APU Primary Port accesses to PCI configuration cycles. APU accesses to PCI Configuration Space use the big endian format. For example, the APU accesses the Command registers as a halfword with PCI Configuration Space offset 0x06 and accesses the Status register as a halfword with PCI Configuration Space offset 0x04.

The PCI_Cfg bit in the PP_Ctrl register enables/disables the APU's ability to drive configuration cycles onto the PCI Bus. When this bit is set, all APU accesses to the Primary Port are driven onto the PCI Bus as configuration read and configuration write cycles. When the PCI_Cfg bit is cleared, all APU accesses to the Primary Port are driven onto the PCI Bus as memory read and memory write cycles.

Figure 9.22 and Figure 9.23 illustrate the timing of PCI configuration cycles driven by the L64364. To initiate the configuration cycle, the L64364 asserts PCI_REQn to arbitrate for the PCI Bus. When the L64364 receives PCI_GNTn, it places the configuration register address on the PCI_AD Bus. Four clock cycles later, the L64364 asserts PCI_FRAMEn and issues a configuration read or write command (PCI_CBEn[3:0] = 0xA or 0xB). The four clock-cycle delay here allows external logic to decode the PCI_AD signals and drive the appropriate PCI_IDSEL signal. The L64364 asserts PCI_IRDYn one clock cycle later and waits for the target to respond. The configuration cycle completes when the target responds by asserting PCI_DEVSELn and PCI_IRDYn or PCI_STOPn. Error cases where the target does not respond correctly are covered in Section 9.5.2, "Master Write Errors" and Section 9.5.4, "Master Read Errors."

9.3 Primary Port Registers

In addition to the Configuration Space registers, three control registers help determine the operation of the PCI Interface and two error registers provide information for error recovery and debugging. They are the XPP_Ctrl register, PP_Ctrl register, PP_SlavePFtch register, PP_Err register, and PP_ErrAddr register. The XPP_Ctrl register (Figure 9.24) is accessible using PCI memory space and not by the APU. It provides PCI host control of PCI interrupts and the L64364 boot process. The PP_Ctrl register (Figure 9.24) limits burst size on the PCI Bus and the Secondary Bus, and controls APU access to the PCI Configuration Space. The PP_SlavePFtch register limits the amount of data prefetched for target

read requests. The PCI_Err register indicates what caused the first received PCI-related error condition (reported by the IntPCIErr interrupt). The PCI_ErrAddr register holds the address associated with the errored access.

For most applications, the L64364 connects to PCI systems with high latency. The PCI Interface optimizes performance by allowing long bursts. To achieve optimum system performance, the burst length on the PCI Bus may need to be fine-tuned with Secondary Bus burst length. The PCI Interface and the Secondary Bus controller integrate maximum burst-size timers which are controlled by the PP_Ctrl register. The PCI Interface maximum burst-size timer works independently from the PCI latency timer. These maximum burst-size timers provide a mechanism for breaking up long packet transfers so the EDMA cell processors or APU do not stall.

9.3.1 XPP_Ctrl Register

This register provides a variety of control functions and is located at PCI Base Address 1 + 0x4010. The register layout is shown in Figure 9.24.

Figure 9.24 XPP_Ctrl Register

7	6	5	4	3	2	1	0
XPP_ APU_Reset	XPP_ BootFault	XPP_APU_W Reset	XPP_ EnInt	XPP_ MbxRxFull	XPP_ MbxRxEmpty	XPP_ MbxTxFull	XPP_ MbxTxEmpty
Default Value and Read/Write Status							
A[1]	0x0				1	0	1
R/W	R	R/W		R			

1. Default dependent on state of SYS_Boot[1:0] pins.

XPP_APU_Reset

PCI Host Reset **7**

When set, XPP_APU_Reset holds all ATMizer II+ modules in the reset state. All modules remain reset until XPP_APU_Reset is cleared. When XPP_APU_Reset clears, the APU initiates the boot sequence.

The default setting of XPP_APU_Reset at initialization depends on the state of the SYS_Boot[1:0] input signals when PCI_RSTn is asserted. This allows the PCI host to control the start of the boot sequence when the

ATMizer II+ chip is booting from Cell Buffer Memory. When booting from EPROM, the boot sequence does not depend on the PCI host; instead, it starts when PCI_RSTn is deasserted. The following table defines the XPP_APU_Reset default values:

SYS_Boot	Boot Source	XPP_APU_Reset Default Value
0b00	Secondary Bus EPROM	0b0
0b01	Not Used	0b1
0b10	Cell Buffer Memory	0b1
0b11	Serial EPROM	0b0

XPP_BootFault

Bus Error Boot Fault **6**

XPP_BootFault sets when a bus error occurs and the APU_Reset bit (bit 31) in the APU_AddrMap register (page 4-99) is set. This bit notifies the PCI host of a possible boot failure due to a bus error that occurred before the exception vectors were initialized.

When XPP_BootFault is set, the APU is prevented from initiating access to the ACI, EDMA, and Scheduler registers. It can still access CBM, Secondary Bus memory, and the PCI Bus.

The boot fault condition can be cleared only by asserting PCI_RSTn.

XPP_APU_WReset

PCI Host Warm Reset **5**

When set, XPP_APU_WReset holds the APU in warm reset. All other modules remain unaffected. When XPP_APU_WReset clears, the APU initiates the boot sequence. The APU_WReset bit in the APU_AddrMap register indicates that the last reset was due to an XPP_APU_WReset.

The main intent of this reset bit is to allow the L64364 to recover from an error condition and keep as much data as possible intact for debugging purposes.

XPP_EnInt **PCI Interrupt Enable** **4**

When set, XPP_EnInt enables assertion of the PCI interrupt signal (PCI_INTn) when the Transmit Mailbox is not empty.

XPP_MbxRxFull

Receive Mailbox Full **3**

The PCI Interface sets XPP_MbxRxFull when the Receive Mailbox is full.

XPP_MbxRxEmpty

Receive Mailbox Empty **2**

The PCI Interface sets XPP_MbxRxEmpty when the Receive Mailbox is empty.

XPP_MbxTxFull

Transmit Mailbox Fulll **1**

The PCI Interface sets XPP_MbxTxFull when the Transmit MailBox is full.

XPP_MbxTxEmpty

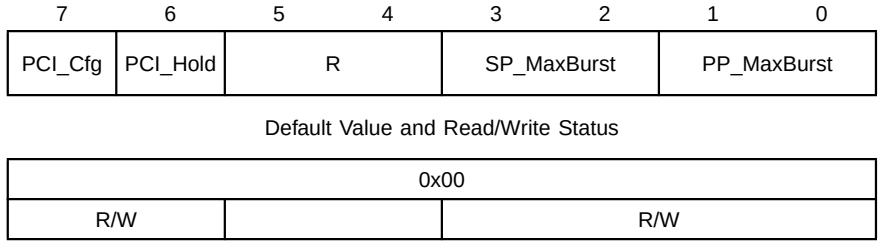
Transmit Mailbox Empty **0**

The PCI Interface sets XPP_MbxTxEmpty when the Transmit MailBox is empty.

9.3.2 PP_Ctrl Register

This register controls access to the PCI Configuration Space and specifies the maximum burst rate for certain transactions. Its format is shown in Figure 9.25. It is located at APU address 0xB800.0403.

Figure 9.25 PP_Ctrl Register



PCI_Cfg **PCI Configuration Select** **7**

When set, APU accesses to the Primary Port are issued as PCI configuration read and configuration write commands. When cleared, APU accesses to the Primary Port are issued as PCI memory read and memory write commands.

PCI_Hold **PCI Target Hold** **6**

When set, all PCI requests, including configuration cycles, targeting the L64364 are issued a retry. This is used to hold off PCI target accesses until boot code has had a chance to initialize the L64364 PCI Configuration registers.

You must initialize the SB_64M bit in the SP_Ctrl register (affects PCI Base Address Register 2) and the PCI Subsystem_ID and Subsystem_Vendor_ID configuration registers to appropriate values before allowing external PCI hosts to access to the L64364 PCI Configuration Space.

R **Reserved** **[5:4]**

Not used in the L64364.

SP_MaxBurst[1:0]

Secondary Port Maximum Burst Length **[3:2]**

SP_MaxBurst[1:0] sets the maximum number of words in an EDMA move command burst or PCI slave write burst. Bit encoding is as follows:

SP_MaxBurst	Burst Length
0b00	Unlimited
0b01	16 32-bit words maximum
0b10	24 32-bit words maximum
0b11	32 32-bit words maximum

PP_MaxBurst[1:0]

Primary Port Maximum Burst Length **[1:0]**

PP_MaxBurst[1:0] sets the maximum number of words in a burst when the EDMA executes a move command. Bit encoding is the same as for SP_MaxBurst[1:0].

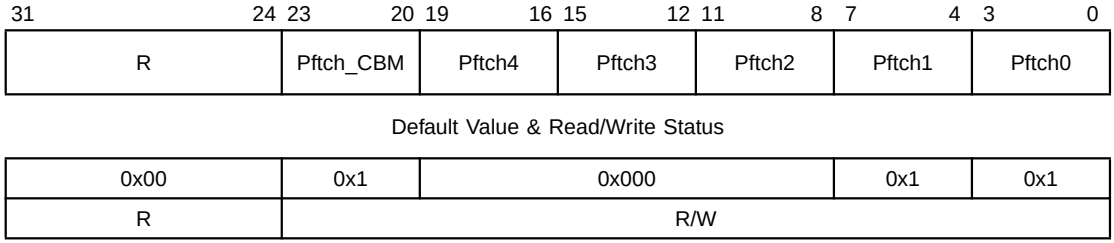
9.3.3 Primary Port Slave Prefetch Register

This register specifies the number of words to be prefetched on a PCI slave access to the Cell Buffer Memory and Secondary Bus. Its format is shown in Figure 9.26. It is located at APU address 0xB800.0410.

The initial byte enables sent with a PCI target read request are used when prefetching the first word of data. All remaining words in the prefetch assume all byte enables active. So, programming a memory

page with a Pftch value of 0x1 allows you to do byte-read accesses to that memory page.

Figure 9.26 PP_SlavePFtch Register



R **Reserved** **[31:24]**
Not used in the L64364.

Pftch_CBM[3:0]
Prefetch from CBM **[23:20]**
Pftch_CBM sets the number of data words to prefetch from CBM when it is the target of a PCI slave access. The bit coding is as follows:

Pftch_CBM[3:0]	Number of Data Words Prefetched
0x0	unlimited[1]
0x1–0xF	1–15

1. Keep the PCI Slave Read FIFO full. This results in prefetching more data than necessary. The PCI Slave Read FIFO is eight words deep.

Pftch4[3:0] **Prefetch from SDRAM** **[19:16]**
Pftch4 sets the number of data words to prefetch from Secondary Bus Page 4 (SDRAM) when it is the target of a PCI slave access. The bit coding is the same as for Pftch_CBM.

Pftch3[3:0] **Prefetch from SSRAM** **[15:12]**
Pftch3 sets the number of data words to prefetch from Secondary Bus Page 3 (SSRAM) when it is the target of a PCI slave access. The bit coding is the same as for Pftch_CBM.

Pftch2[3:0] **Prefetch from 32-Bit SRAM** **[11:8]**
Pftch2 sets the number of data words to prefetch from Secondary Bus Page 2 (32-bit SRAM) when it is the target of a PCI slave access. The bit coding is the same as for Pftch_CBM.

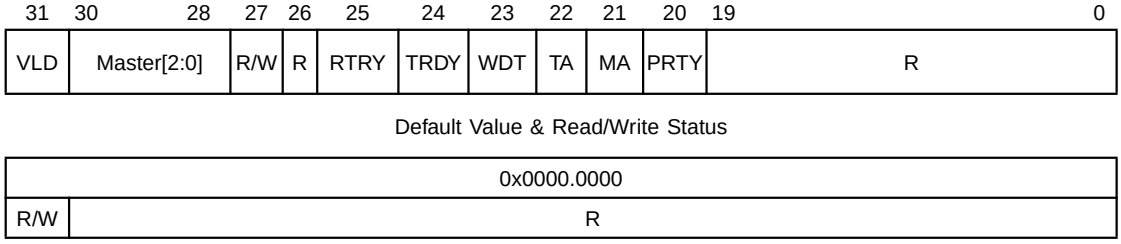
Pftch1[3:0] **Prefetch from 8-Bit PHY** **[7:4]**
Pftch1 sets the number of data words to prefetch from Secondary Bus Page 1 (8-bit PHY) when it is the target of a PCI slave access. The bit coding is the same as for Pftch_CBM.

Pftch0[3:0] **Prefetch from 8-Bit EPROM/SRAM** **[3:0]**
Pftch0 sets the number of data words to prefetch from Secondary Bus Page 0 (8-bit EPROM/SRAM) when it is the target of a PCI slave access. The bit coding is the same as for Pftch_CBM.

9.3.4 Primary Port Error Register

This register contains information on the type of error and the offending master. The PP_Err register captures information on the first error received and contains valid data when the VLD bit is set. The APU re-enables the capturing of future errors by clearing the VLD bit. The register format is shown in Figure 9.27. It is located at APU address 0xB800.0420.

Figure 9.27 PP_Err Register



VLD **Primary Port Error Valid Bit** **31**
VLD is set when the PP_Err register has captured data pertaining to a Primary Port error condition. VLD is cleared by writing a 1 to the bit position. Clearing the VLD bit enables the capturing of data pertaining to the next error condition.

Master[2:0] **Primary Port Master Field** **[30:28]**
These bits indicate the master that caused the Primary Port error condition as follows:

Master[2:0]	Master of Access
0b000	APU
0b001	TMU
0b010	EDMA
0b011	MOVE
0b100	External Host Base Address 2 Decode
0b101	External Host Base Address 1 Decode
0b110	Reserved
0b111	Reserved

R/W **Read/Write Indicator** **27**
When the R/W bit is set, the Primary Port error occurred on a read access. When the bit is cleared, the Primary Port error occurred on a write access.

R **Reserved** **26**
Not used in the L64364.

RTRY **Retry Timer Time-Out** **25**
When set, indicates that the PCI Retry Timer timed out. See the Retry_Timer register description in Section 9.2.19. This bit is cleared at reset.

TRDY **TRDY Timer Time-Out** **24**
When set, indicates that the PCI TRDY Timer timed out. See the TRDY_Timer register description in Section 9.2.18. This bit is cleared at reset.

WDT **WatchDog Time-Out** **23**
When set, indicates that the access caused an APU WatchDog time-out. This bit is cleared at reset.

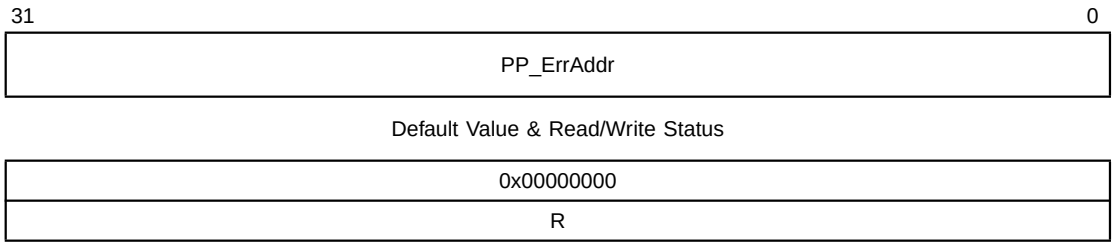
TA **Target Abort** **22**
When set, indicates that the Primary Port detected a PCI target abort. This bit is cleared at reset.

MA	**Master Abort** When set, indicates that the Primary Port detected a PCI master abort. This bit is cleared at reset.	**21**
PRTY	**Parity Error** When set, indicates that the Primary Port detected a PCI parity error. This bit is cleared at reset.	**20**
R	**Reserved** Not used in the L64364.	**[19:0]**

9.3.5 Primary Port Error Address Register

This register contains the address associated with an errored access. Its format is shown in Figure 9.28. It is located at APU address 0xB800.0424.

Figure 9.28 PP_ErrAddr Register



PP_ErrAddr	**Primary Port Error Address** Primary Port address that was being accessed when the error occurred. Due to pipelining delays associated with the PCI read and write FIFOs, the PP_ErrAddr register provides the initial address of burst accesses that caused a bus error.	**[31:0]**

9.4 PCI Slave Transactions

PCI Base Address Register 1 (page 9-12) supports access to the L64364 Cell Buffer Memory, Mailbox FIFO, XPP_Ctrl register, and APU hardware registers. Table 9.2 provides a memory map for this address range.

Table 9.2 ATMizer II+ Chip External Memory Map

PCI Memory	Module	Size
0x0000–0x0FFF	Cell Buffer Memory	4 Kbytes
0x1000–0x3FFF	Reserved[1]	12 Kbytes
0x4000–0x400F	Mailbox FIFO	16 Bytes
0x4010	XPP_Control register	1 Byte
0x4011–0x6FFF	Reserved[2]	< 16 Kbytes
0x7000–0x7FFF	L64364 registers[3]	< 16 Kbytes

1. PCI slave writes to this reserved area are completed normally but data is ignored. PCI slave reads of this reserved area wrap to the corresponding location in Cell Buffer Memory.
2. PCI slave writes to this reserved area are completed normally but data is ignored. PCI slave reads of this reserved area return all 1s.
3. See Appendix A for a summary of L64364 hardware registers.

Note: Burst reads of the L64364 Mailbox FIFO and registers are not supported. Burst reads/writes to Cell Buffer Memory and burst writes to the Receive Mailbox FIFO are permitted.

PCI Base Address register 2 (page 9-13) maps the ATMizer's local memory into PCI memory space. Refer to Table 10.1 on page 10-3 for the local memory address map.

9.4.1 Mailbox

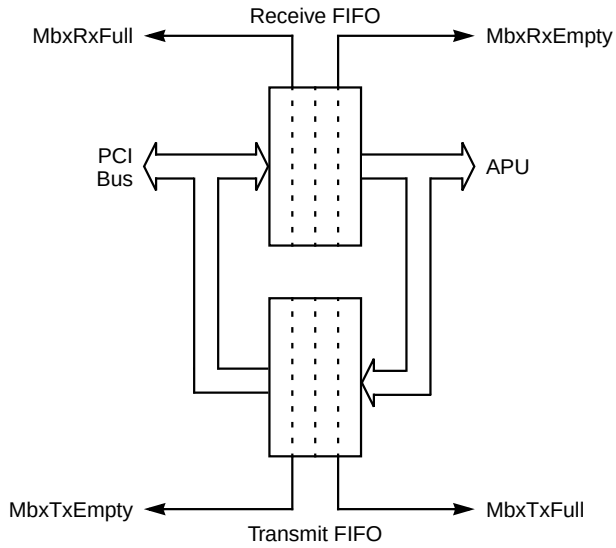
The Mailbox consists of two FIFOs, each are 32-bits wide and 4-words deep. The Mailbox assures faster communication between the APU and an external PCI Bus master. One FIFO (the Transmit FIFO) provides for APU-to-PCI host messaging, and the other FIFO (the Receive FIFO) for PCI host-to-APU messaging. From the APU side, the Mailbox appears to be a memory-mapped hardware register, called APU_MailBox. From the external PCI Bus master, it is mapped above Cell Buffer Memory.

The PCI master may access the Mailbox by word read or write at address offsets 0x4000, 0x4004, 0x4008, or 0x400C. The behavior of the Mailbox is independent of the address by which it is accessed. All PCI slave reads of the Transmit FIFO must be single word transfers. Burst transfers are supported on PCI slave writes to the Receive FIFO.

The APU writes to the Transmit FIFO at address 0xB800.0408 and reads the Receive FIFO at address 0xB800.0404. APU writes to the Receive FIFO or APU reads from the Transmit FIFO are not supported.

Figure 9.29 shows the implementation of the Mailboxes. The MbxTxEmpty signal generates the PCI_INTn signal. The PCI Interface asserts PCI_INTn when enabled in the XPP_Ctrl register (page 9-22) and the Transmit FIFO is not empty. External controllers must isolate the Mailbox from events that cause overflow of the FIFO.

Figure 9.29 Mailbox Registers



If an external PCI Bus master reads an empty Transmit FIFO, it receives a 0 value. If a PCI Bus master attempts to write to a full Receive FIFO, the PCI Interface drops the written data and generates a nonvectored interrupt to the APU. The external master must prevent overflow of the FIFO. Note that Mailbox status signals are visible to the PCI Bus master in the XPP_Ctrl register.

If the APU reads an empty Receive FIFO, it receives a 0 value. If the APU attempts to write to a full Transmit FIFO, the APU stalls until the FIFO becomes nonempty.

The MbxRxEmpty signal is internally connected to the APU Vectored Interrupt IntRxMbx (Receive FIFO not empty) described in Section 4.8.2, "External Vectored Interrupt Sources." In addition, the IntRxMbx interrupt

sets the APU_RxMbx bit and the MbxTxFull bit sets the APU_MbxFull bit in the APU_Status register (page 4-108). The MbxRxFull signal is used to create IntRxMbxOvr, nonvectored interrupt 2, as shown in Section 4.8.1, “External Nonvectored Interrupts.”

9.4.2 PCI Slave Write Timing

Figure 9.30, Figure 9.31, and Figure 9.32 illustrate PCI slave write timing. Figure 9.30 illustrates a slave write transfer of four words. When the ATMizer II+ chip’s Slave Write FIFO becomes full, the L64364 disconnects as shown in Figure 9.31. Figure 9.32 illustrates timing when the PCI Interface detects data parity errors (PCI_PERRn).

To initiate a PCI slave write transfer (Figure 9.30), an external master asserts PCI_FRAMEn and issues a write command to the L64364 (PCI_CBE[3:0] = 0x7 or 0xF) with an address in the range specified by one of the Base Address registers (page 9-12). The external bus master also asserts the PCI Initiator Ready (PCI_IRDYn) signal to indicate when there is valid data on the PCI Bus (PCI_AD[31:0]). The PCI_CBEn lines then become byte enables.

In response to PCI_FRAMEn, the L64364 asserts PCI Device Select (PCI_DEVSELn) two clock edges later (medium timing, see DEVSEL[1:0] on page 9-9).

The L64364 asserts the PCI Target Ready (PCI_TRDYn) signal when it can accept write data from the PCI Bus. The condition of the Slave Write FIFO determines when the L64364 can accept data. If the Slave Write FIFO is full, the L64364 continues to retry the transaction until the FIFO becomes not full.

If the Slave Write FIFO is full (during a burst transaction), the L64364 disconnects from the PCI Bus by asserting PCI_STOPn as shown in Figure 9.31.

If the PCI Interface attempts a slave write to a Secondary Bus Memory page that is not enabled, the transfer is not aborted. Instead, the transfer finishes normally, and the Secondary Bus Controller issues a Secondary Bus Address error which causes an Int_SBErr interrupt (nonvectored interrupt 4).

Figure 9.30 Slave Write Timing

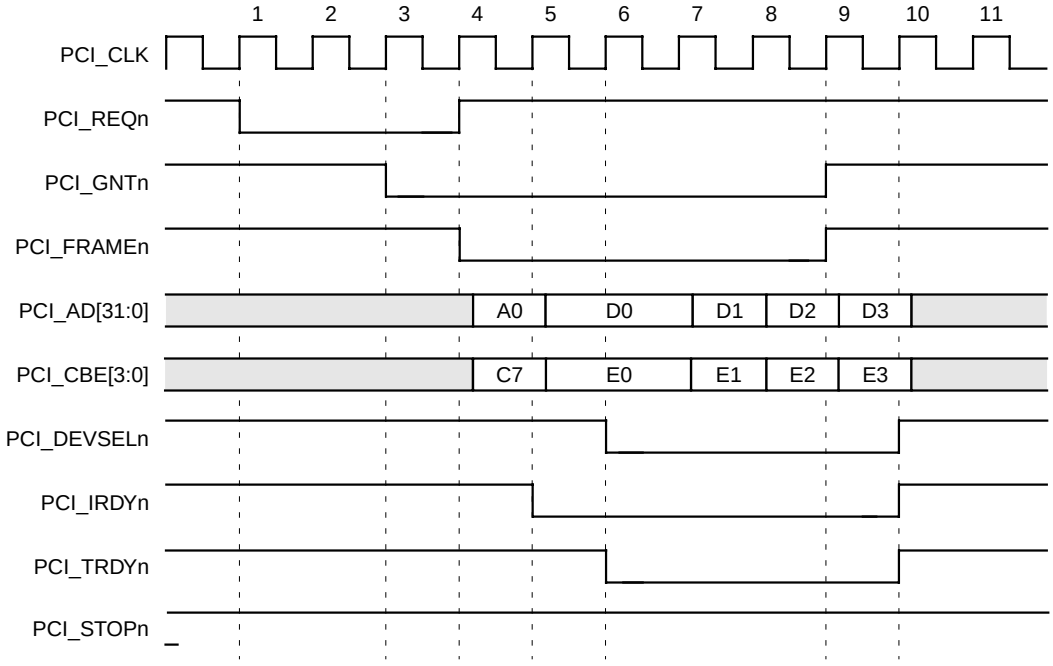
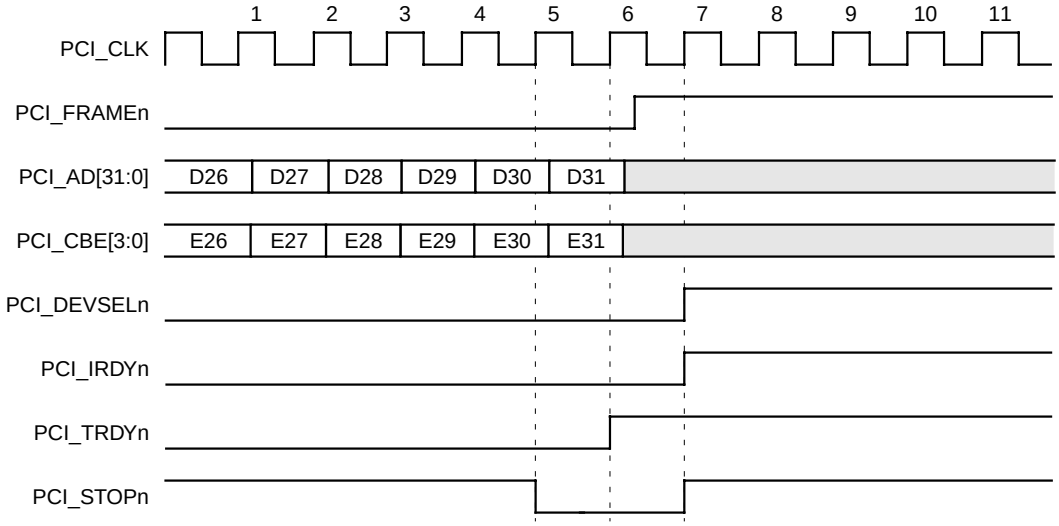
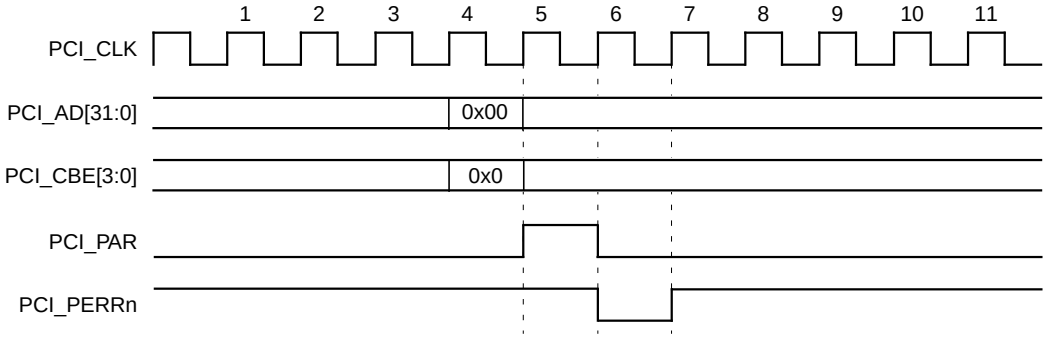


Figure 9.31 Slave Write Stop Timing



If the Parity Error Response bit (PER) in the Command register (page 9-7) is set, the PCI Interface asserts PCI_PERRn when it detects a data parity error (see Figure 9.32). If a data parity error is detected during a slave write operation, the error is reported to the APU as IntPCIErr, nonvectored interrupt 5.

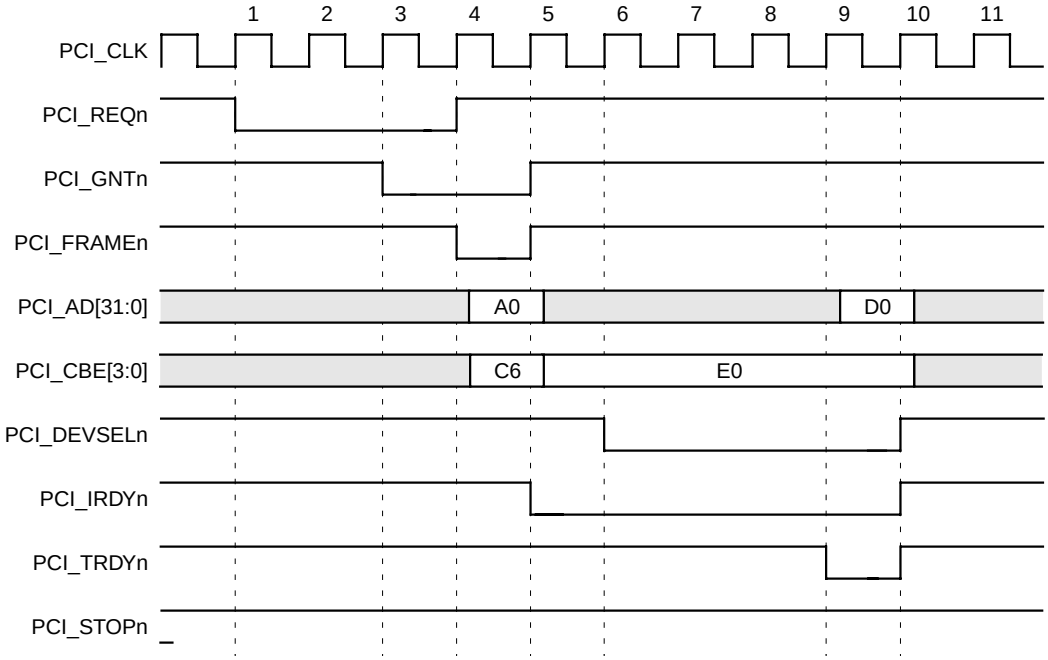
Figure 9.32 Parity Error Timing



9.4.3 PCI Slave Read Timing

Figure 9.33 illustrates the PCI slave read timing. To initiate a PCI slave read transfer, an external master asserts PCI_FRAMEn and a read command (PCI_CBE[3:0] = 0x6, 0xC, or 0xE) with an address in the range specified by one of the Base Address registers. The external bus master also asserts the PCI Initiator Ready (PCI_IRDYn) signal to indicate when it can accept data from the PCI Bus (PCI_AD[31:0]). In response to PCI_FRAMEn, the L64364 asserts PCI_DEVSELn two clock edges later (medium DEVSEL timing).

Figure 9.33 PCI Slave Read Timing



The L64364 asserts the PCI Target Ready (PCI_TRDYn) signal to indicate that there is data available in the Slave Read FIFO. If data is not available within 16 clocks, the L64364 asserts PCI_STOPn and retries the transaction. In this case, all subsequent slave read commands to other addresses are retried until the transaction completes. If the original PCI master does not retry the transaction within 2^{16} PCI clock cycles, the PCI Interface flushes the slave read data.

If the PCI Bus master attempts a slave read of a Secondary Bus Memory page that is not enabled, the transfer is not aborted. Instead, the transfer finishes with 0xFFFF.FFFF returned as data and the Secondary Bus Controller issues a Secondary Bus Address error. This error then generates an Int_SBErr interrupt (nonvectored interrupt 4) to the APU.

9.4.4 PCI Slave Errors

The following error conditions can occur during a PCI slave access:

- Data parity error
- Address parity error

For data parity errors, the L64364 completes the data transfer and, if enabled by the PER bit in the Command register, asserts PCI_PERRn on PCI slave writes and generates an IntPCIErr interrupt to the APU.

For address parity errors, the L64364 claims the transfer cycle and terminates it with target abort. This occurs regardless of the state of the SERR bit in the Command register. However, if the SERR bit is set, the L64364 also asserts PCI_SERRn.

9.5 PCI Master Transactions

The L64364 requests a PCI master transaction when the APU or EDMA accesses Primary Memory space. As PCI Bus master, the L64364 can generate the following commands:

PCI_CBE[3:0]	Command
0b0110	Memory Read
0b0111	Memory Write
0b1100	Memory Read Multiple
0b1110	Memory Read Line

The four bus masters, APU, EDMA Move processor, EDMA TxCell Processor, and EDMA RxCell Processor, all arbitrate for PCI Master transactions by using a round-robin arbitration technique.

Note: L64364 masters must not initiate a PCI cycle with an address that targets the same L64364's PCI slave interface. Should this happen, the L64364's PCI slave interface tries to respond to the request. This results in either bad data being transferred, or an L64364 PCI interface lock up. The results are not predictable.

9.5.1 PCI Master Write Timing

Figure 9.34 and Figure 9.35 illustrate master write timing. Figure 9.34 contains a master write burst of four words. The same transfer is initiated in Figure 9.35, but then it is terminated by a target disconnect.

Figure 9.34 Master Write Timing

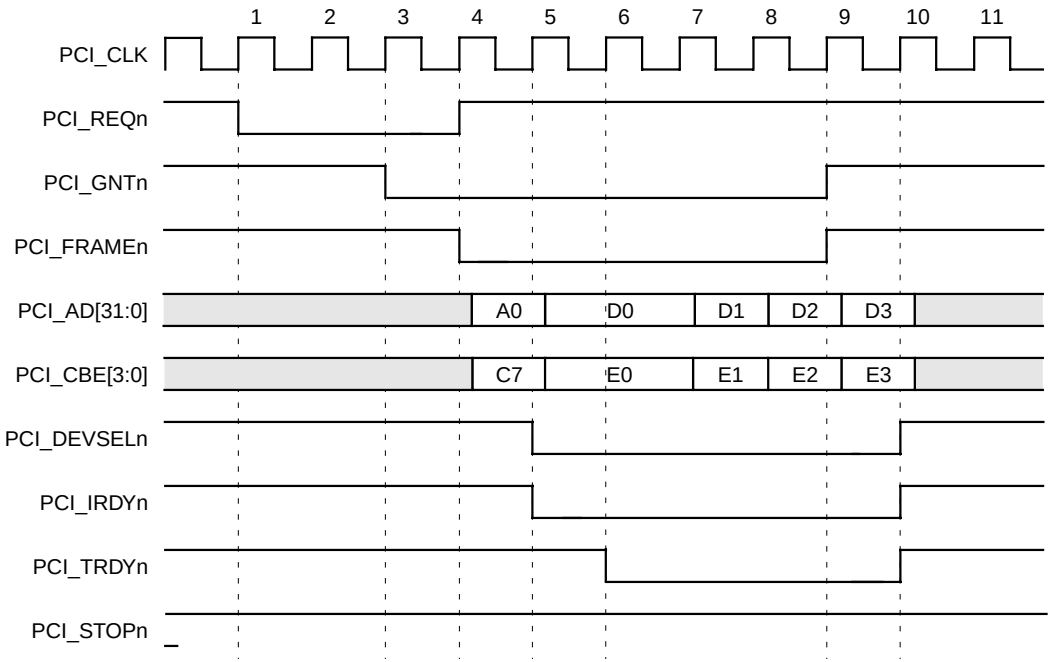
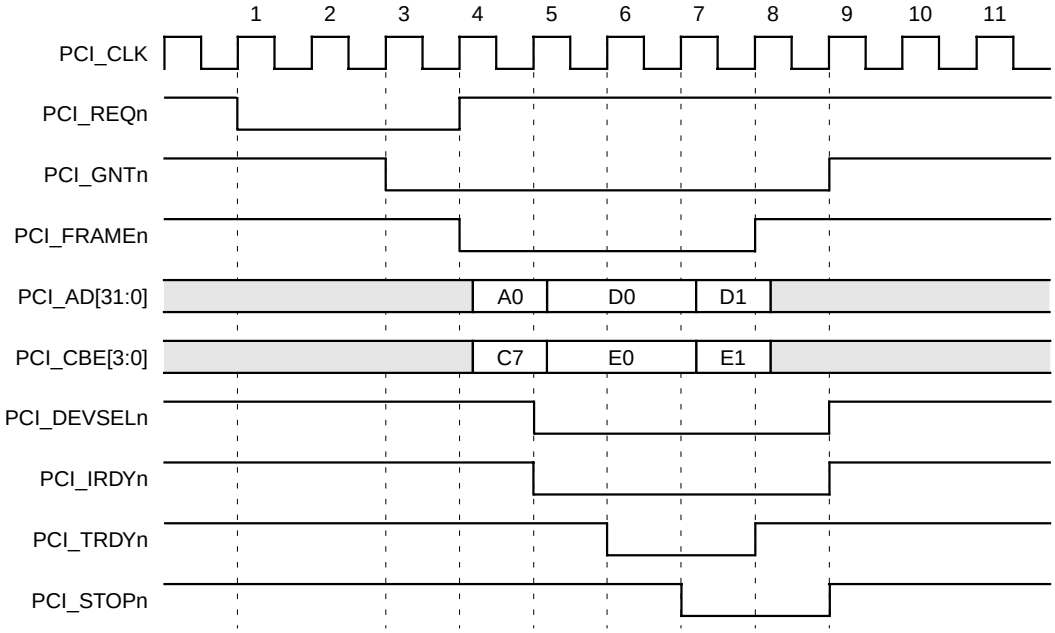


Figure 9.35 Master Write Stop Timing



The ATMizer's Primary Port module writes data to the Master Write FIFO when the APU or EDMA issues a write bus request to the Primary Memory space. When the Master Write FIFO becomes nonempty, the PCI Interface asserts `PCI_REQn`. For burst transactions, the Primary Port module continues to fill the Master Write FIFO until one of the following events occur:

- The Master Write FIFO becomes full.
- The maximum burst size specified by the `PP_MaxBurst` bits in the PP_Ctrl register (page 9-24) is reached.
- The burst is completed.

Case 1 can occur only when the EDMA processes a `move` command to PCI Memory. As explained in Section 5.3.4, "Move Command," the command transfers a block of data between the Secondary Port memory (local memory) and the Primary Port memory (PCI memory). The EDMA Move Processor retains ownership of the PCI Interface but releases the Secondary Bus to allow interleaved accesses by the EDMA Cell Processors or APU. When the Master Write FIFO reaches half full, the

Move Processor requests the Secondary Bus and continues the move transfer.

Case 2 occurs only when the EDMA is processing a move command. In this case, the Primary Port allows other bus masters (APU and EDMA Cell processors) to access the PCI Master Write FIFO.

The PCI Interface terminates the PCI transaction (normal completion) when the Master Write FIFO becomes empty. If this occurs before completion of the intended burst transfer, the PCI Interface requests the PCI Bus again and continues until the transaction finishes.

9.5.2 Master Write Errors

The following error conditions can occur during a master write transaction:

- Master abort
- Target abort
- Parity error

Master abort occurs if PCI_DEVSELn is not returned within six PCI clocks or PCI_TRDYn is not returned within 128 PCI clocks after a memory write command. Target abort is an abnormal termination requested by the selected target. The target reports parity errors to the L64364 on the PCI_PERRn pin.

Target abort and master abort cause the PCI Interface to signal a bus error to the current bus master. This in turn may cause one or more nonvectored interrupts to be generated to the APU. If the APU was the bus master, exception code 6 (bus error) is generated. If the EDMA was the bus master, an Int_EDMA_BusErr interrupt (nonvectored interrupt 0) may be generated to the APU.

Reporting master write errors back to the bus master (EDMA or APU) is not reliable due to the pipelining of PCI master transactions. Therefore, the error is also reported to the APU as IntPCIErr, nonvectored interrupt 5.

For data parity errors, the L64364 completes the data transfer and, if enabled by the PER bit in the Command register, generates an IntPCIErr interrupt (nonvectored interrupt 5) to the APU.

9.5.3 PCI Master Read Timing

Figure 9.36 through Figure 9.38 illustrate master read timing. Figure 9.36 shows a four-word, master read, burst transfer. Figure 9.37 shows the same transfer being initiated but here the transfer terminates due to target disconnect. Figure 9.38 illustrates PCI_PERRn when a data parity error occurs.

The Primary Port initiates a PCI read transaction when the APU or EDMA executes a read operation to Primary Memory space. The PCI command depends on the transaction size and the value programmed in the PCI Cache Line Size register.

- If the Cache Line Size register is programmed to zero, the L64364 always generates the Memory Read command.
- If the Cache Line Size register is programmed with a nonzero value and the requesting master requests a data block whose size is less than the Cache Line Size register, the L64364 generates a Memory Read command.
- If the Cache Line Size register is programmed with a nonzero value, the requesting master requests a data block whose size is greater than or equal to the Cache Line Size register, and the data block starts on a cache line boundary, the L64364 generates a Memory Read Line command.
- If the Cache Line Size register is programmed with a nonzero value, the requesting master requests a data block whose size is greater than or equal to the Cache Line Size register, and the data block does not start on a cache line boundary, the L64364 generates a Memory Read Multiple command.

Figure 9.36 PCI Master Read Timing

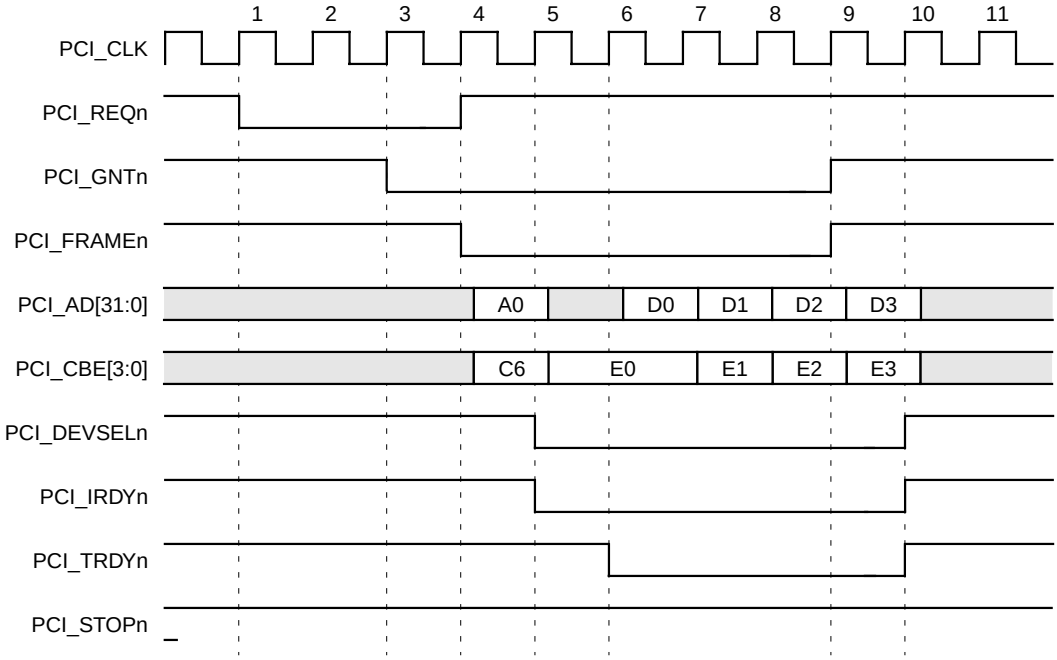


Figure 9.37 Master Read Stop Timing

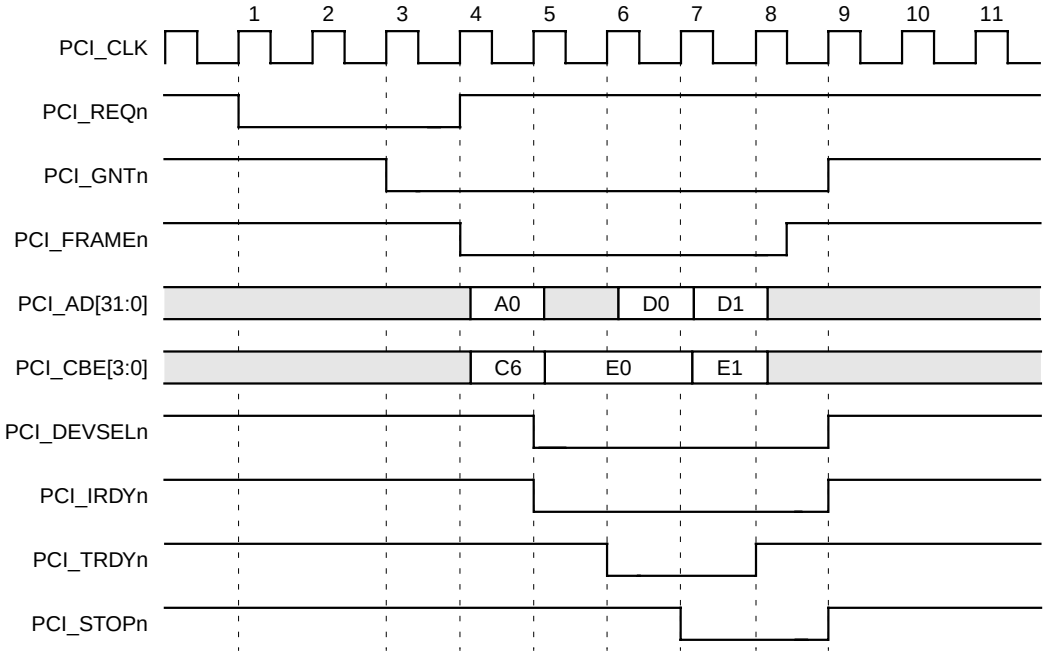
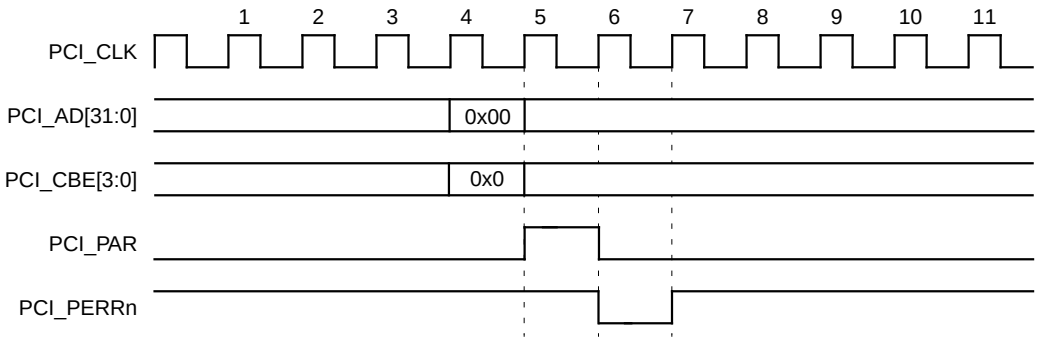


Figure 9.38 Master Read Error Timing



9.5.4 Master Read Errors

The following error conditions can occur during a Master Read transaction:

- Master abort
- Target abort
- Parity error

Master abort occurs when PCI_DEVSELn is not returned within six PCI clocks or PCI_TRDYn is not returned within 128 PCI clocks after a memory read, memory read line, or memory read multiple command. Target abort is an abnormal termination requested by the selected target. When a data parity error is detected, the PCI Interface generates a parity error only if the Parity Error Response bit (PER bit) in the Command register is set.

Target abort and master abort cause the PCI Interface to signal a bus error to the current bus master. This in turn may cause one or more nonvectored interrupts to be generated to the APU. The IntPCIErr interrupt (nonvectored interrupt 5) is always generated for target abort and master abort. If the APU was the bus master, exception code 6 (bus error) is generated. If the EDMA was the bus master, an Int_EDMA_BusErr interrupt (nonvectored interrupt 0) may be generated to the APU.

For data parity errors, the L64364 completes the data transfer and if enabled by the PER bit in the Command register, asserts PCI_PERRn and generates an IntPCIErr interrupt to the APU.

Data parity errors are reported to the APU as a nonvectored interrupt 5.

9.6 Balancing Bus Usage

When transferring data between the PCI and Secondary Memory, the ATMizer II+ chip handles high latency by balancing bus usage between the PCI Bus and the Secondary Bus. The PCI Bus tends to be a high-latency bus, so it is better to maximize burst length. The Secondary Bus has real time requirements from the APU, EDMA, and Scheduler. This causes high Secondary Bus latency.

The sections that follow summarize the PCI FIFO protocol used to balance PCI and Secondary Bus usage. PCI FIFO size and thresholds are optimized assuming a 66 MHz or greater, zero wait state, Secondary Memory. Lower performance Secondary Memories are also supported, but can cause additional disconnects during burst transactions. Refer to Figure 9.1 at the beginning of this chapter during the following discussions.

9.6.1 Master Write

This section summarizes the steps used to perform a master write transaction initiated by the EDMA.

The EDMA Move Processor requests the Secondary Bus and requests to read `MoveCount` bytes. `MoveCount` is specified in the EDMA_MoveCount register. When the first word is read, the EDMA requests the Primary Port to write `MoveCount` bytes.

When the Primary Port request is granted, the EDMA Move Processor transfers data from the Secondary Bus Controller to the PCI Master Write FIFO. If the PCI Master Write FIFO becomes full, the EDMA releases the Secondary Bus.

When the PCI Master Write FIFO drains to half full, the EDMA Move Processor again requests the Secondary Bus. If the Secondary Bus request is granted before the PCI Master FIFO becomes empty, the operation continues.

When the PCI Master FIFO becomes empty, the PCI Interface disconnects from the PCI Bus. If the PCI Interface disconnects from the PCI Bus before `MoveCount` bytes are transferred, it requests the bus again as soon as the PCI Master FIFO becomes nonempty to continue the transaction.

9.6.2 Master Read

This section summarizes the steps used to perform a master read transaction initiated by the EDMA.

The EDMA Move Processor requests to read MoveCount bytes from the Primary Port. This causes the PCI Interface to request the PCI Bus. The EDMA Move Processor is stalled but it owns the Primary Port until the PCI Master Read FIFO fills to 16 words (half full).

When the PCI Master Read FIFO reaches half full or the PCI read completes, the EDMA Move Processor requests the Secondary Bus. When the Secondary Bus request is granted, the EDMA Move Processor transfers data to the Secondary Bus until the PCI Master Read FIFO becomes empty or MoveCount bytes are transferred.

When the PCI Master Read FIFO becomes empty, the EDMA releases the Secondary Bus. If MoveCount bytes have not been transferred, the operation continues when the Master Read FIFO reaches half full. If after this occurs, the PCI Read FIFO fills due to Secondary Bus latency, the PCI Interface disconnects from the PCI Bus. If the PCI Interface disconnects from the PCI Bus before MoveCount bytes are transferred, it requests the bus again as soon as the PCI Master FIFO becomes nonempty.

9.6.3 Slave Write

This section summarizes the steps used to perform a slave write transaction.

When the PCI Slave Write FIFO becomes nonempty, the PCI Interface requests the Secondary Bus. The Secondary Bus Controller transfers the slave write data to Secondary Memory. If the PCI Slave Write FIFO becomes full due to Secondary Bus latency, the PCI Interface disconnects from the PCI Bus.

When the PCI Slave Write FIFO becomes empty, the PCI Interface releases the Secondary Bus. The transaction continues when the PCI Slave Write FIFO becomes not empty.

9.6.4 Slave Read

The PCI Interface requests the Secondary Bus in response to a slave read command from a PCI master. When the Secondary Bus is available, data is transferred to the PCI Bus through the PCI Slave Read FIFO. The Secondary Bus Controller prefetches data until the programmed prefetch limit is reached (see the PP_SlavePftch register, page 9-26), the Slave Read FIFO becomes full, or the PCI master terminates the operation.

Chapter 10
Secondary Bus Memory Controller

This chapter describes the Secondary Bus Memory Controller (SBC), the L64364's interface to local memory.

This chapter includes the following sections:

- Section 10.1, "Overview," page 10-2
- Section 10.2, "SBC Configuration," page 10-3
- Section 10.3, "Secondary Bus Performance Considerations," page 10-11
- Section 10.4, "SDRAM Controller," page 10-14
- Section 10.5, "SSRAM Controller," page 10-25
- Section 10.6, "32-Bit SRAM/EPROM Controller," page 10-28
- Section 10.7, "PHY Controller," page 10-32
- Section 10.8, "8-Bit SRAM/EPROM Controller," page 10-36
- Section 10.9, "External Bus Masters," page 10-40
- Section 10.10, "Error Reporting," page 10-41

Important: Registers and fields labeled "Reserved" are don't cares. Descriptor bits and fields labeled "Reserved" must not be modified.

10.1 Overview

The SBC provides a seamless interface to local memory. It contains the control logic necessary to directly connect synchronous DRAM, synchronous SRAM, asynchronous SRAM, EPROM, and some physical layer devices to the L64364.

The controller supports both 32-bit and 8-bit wide devices. The 8-bit capability enables connecting byte-wide EPROMs that can contain the APU boot program and a target debugger. A byte-wide interface can also be used to connect to external physical layer devices, such as framers.

An integrated wait-state generator allows direct connection to slower asynchronous SRAM, PHY, and EPROM devices.

Both internal and external PCI Bus masters can access secondary memory. The internal PCI Bus masters include the APU, EDMA, and the Scheduler. A simple bus request/grant arbiter allows connection of external bus masters directly on the Secondary Bus.

Access to the SBC is gained through a round-robin arbitration technique. The participants include:

- PCI Slave Interface
- APU
- EDMA TxCell Processor
- EDMA RxCell Processor
- EDMA Move Processor
- Scheduler
- External Secondary Bus master

Starving of Secondary Bus agents can be minimized by limiting burst size with `SP_MaxBurst[1:0]` in the PP_Ctrl register (page 9-24).

The SBC has five memory pages as shown in Table 10.1 and Table 10.2, one for each external memory device. The SBC uses the memory page address to select the control sequence appropriate for a particular external memory device.

Table 10.1 16 Mbyte Secondary Bus Memory Map[1]

Start Address	End Address	Size	Device Type	Bus Size
0x0000.0000	0x000F.FFFF	1 Mbyte	EPROM/SRAM	8
0x0020.0000	0x002F.FFFF	1 Mbyte	PHY	8
0x0040.0000	0x005F.FFFF	2 Mbytes	EPROM/SRAM	32
0x0060.0000	0x007F.FFFF	2 Mbytes	SSRAM	32
0x0080.0000	0x00FF.FFFF	8 Mbytes	SDRAM	32

1. Selected when the SB_64M bit in the SP_CTRL register is cleared.

Table 10.2 64 Mbyte Secondary Bus Memory Map[1]

Start Address	End Address	Size	Device Type	Bus Size
0x0000.0000	0x000F.FFFF	1 Mbyte	EPROM/SRAM	8
0x0020.0000	0x002F.FFFF	1 Mbyte	PHY	8
0x0040.0000	0x007F.FFFF	4 Mbytes	EPROM/SRAM	32
0x0080.0000	0x00BF.FFFF	4 Mbytes	SSRAM	32
0x0200.0000	0x03FF.FFFF	32 Mbytes	SDRAM	32

1. Selected when the SB_64M bit in the SP_CTRL register is set.

When accessing asynchronous SRAM, EPROM, or peripheral devices, bus cycle timing is controlled either with an integrated wait-state generator or the data-valid input signal, SB_RDYn.

10.2 SBC Configuration

The SBC can be configured to support all five local memory pages concurrently. Electrical characteristics limit the number of devices that directly connect to the ATMizer II+ chip. Typically, only one bank of SDRAM, SSRAM, or SRAM are directly connected to the L64364 ATMizer II+ chip while EPROM and PHY devices are buffered with a transceiver.

10.2.1 SP_Ctrl Register

The SP_Ctrl register, shown in Figure 10.1, enables the active local memory pages and selects the number of wait states for EPROM, SRAM, and PHY devices. The APU accesses the SP_Ctrl register at memory address 0xB800.0800.

Figure 10.1 SP_Ctrl Register

31 27	26	25	24 18	17 16	15 14	13 12	11 8	7 4	3 0
SB_EnPage [4:0]	SB_ DSL3	SB_ 64M	R	SB_DSL2 [1:0]	SB_DSL1 [1:0]	SB_DSL0 [1:0]	SB_Wait2 [3:0]	SB_Wait1 [3:0]	SB_Wait0 [3:0]
Reset Value & Read/Write Status									
0x1F	0x0	0x0	0x00	0x3F			0xE/0xF[1]		
R/W			R	R/W					

1. The reset values of SB_Wait2, SB_Wait1, and SB_Wait0 depend on the state of SB_RDYn while PCI_RSTn is asserted. If SB_RDYn is a '1,' then the SB_Wait fields reset to 0xE; if SB_RDYn is a '0,' then the SB_Wait fields reset to 0xF.

SB_EnPage[4:0]

Secondary Bus Page Enables [31: 27]

SB_EnPage[4:0] enable the corresponding local memory pages (see table below). An APU or PCI access to a local memory page that is not enabled will cause an SBC address error which asserts nonvectored interrupt 4 (IntSBErr) exception to the APU.

Bit #	Bit Name	Local Memory Page
31	SB_EnPage[4]	SDRAM
30	SB_EnPage[3]	SSRAM
29	SB_EnPage[2]	32-bit SRAM/EPROM
28	SB_EnPage[1]	8-bit PHY
27	SB_EnPage[0]	8-bit SRAM/EPROM

SB_DSL3 **Secondary Bus SSRAM Deselect Timing 26**

SB_DSL3 is used to control deselect timing in the SSRAM page. SB_DSL3 should be set when using single-cycle deselect SSRAM devices and should be cleared when using double-cycle deselect SSRAM devices.

SB_64M **Secondary Bus Memory Map Size Select** **25**
SB_64M cleared selects the 16 Mbyte Secondary Bus Memory Map. SB_64M set selects the 64 Mbyte Secondary Bus Memory Map. See Table 10.1 and Table 10.2.

R **Reserved** **[24:18]**
Not used in the L64364.

SB_DSL2[1:0] **Secondary Bus Page 2 Deselect Timing** **[17:16]**
SB_DSL2 selects the number of clock delays inserted from the deassertion of SB_PCSn[2] (see page 3-10) to the assertion of a new SB_PCSn. The number of clock delays is coded as follows:

SB_DSL2[1:0]	Clock Delay
0b00	0
0b01	1
0b10	2
0b11	3

SB_DSL1[1:0] **Secondary Bus Page 1 Deselect Timing** **[15:14]**
SB_DSL1 selects the number of clock delays inserted from the deassertion of SB_PCSn[1] (see page 3-10) to the assertion of a new SB_PCSn. Bit coding is the same as SB_DSL2.

SB_DSL0[1:0] **Secondary Bus Page 0 Deselect Timing** **[13:12]**
SB_DSL0 selects the number of clock delays inserted from the deassertion of SB_PCSn[0] (see page 3-10) to the assertion of a new SB_PCSn. Bit coding is the same as SB_DSL2.

SB_Wait2[3:0] **Secondary Bus Page 2 Wait States** **[11:8]**
SB_Wait2[3:0] selects the number of wait states inserted in read/write accesses from/to the 32-bit SRAM/EPROM page. The number of wait states is encoded as follows:

SB_Wait2[3:0]	Wait States
0x0–0xE	0–14
0xF	SB_RDYn

When SB_Wait2[3:0] = 0xF, the SB_RDYn (see page 3-10) input is used to terminate the data cycle. SB_RDYn is an asynchronous input and is resynchronized to SB_CLKO before it is used.

When in this mode, if SB_RDYn is not asserted within 64 system clocks, an SBC address error is generated.

SB_Wait1[3:0] Secondary Bus Page 1 Wait States [7:4]

SB_Wait1[3:0] selects the number of wait states inserted in read/write accesses to the PHY page. SB_Wait1 uses the same bit encoding as SB_Wait2.

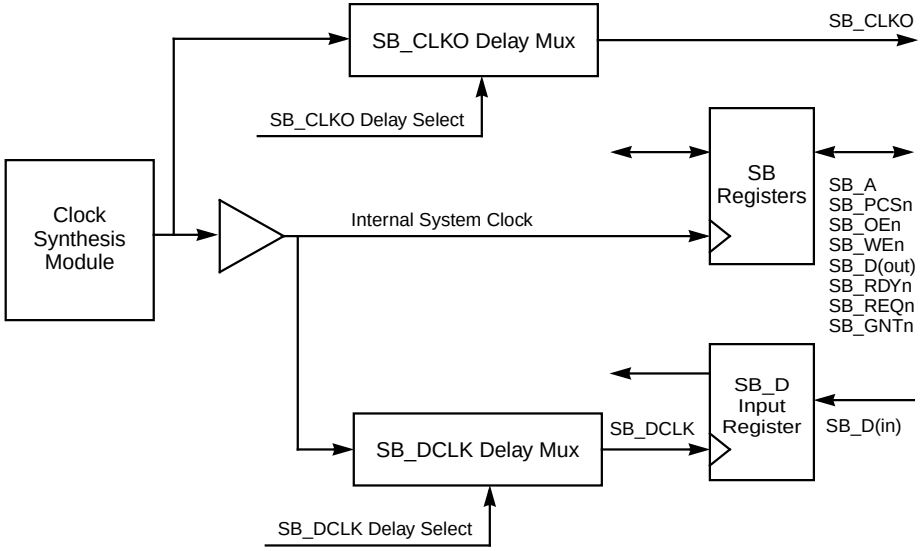
SB_Wait0[3:0] Secondary Bus Page 0 Wait States [3:0]

SB_Wait0[3:0] selects the number of wait states inserted in read/write accesses to the 8-bit SRAM/EPROM page. SB_Wait0 uses the same bit encoding as SB_Wait2.

10.2.2 Secondary Bus Clock Control Register

The Secondary Bus Clock Control register allows you to tune the SB_CLKO output clock and the SB_D[31:0] input read clock (refer to Figure 10.2) to the internal system clock as needed by your system design. The internal system clock is the base clock for Secondary Bus Control logic.

Figure 10.2 SB Clock Relationships



The format of the SB Clock Control register is shown in Figure 10.3 and its fields are described following the figure. The register is accessible at address 0xB800.0810.

Figure 10.3 Secondary Bus Clock Control Register

31 … 11	10 … 8	7 … 3	2 … 0
R	SB_DCLK delay	R	SB_CLKO delay
0x0000			0x4
	R/W		R/W

Reset Value & Read/Write Status

R **Reserved** **[31:11]**
Not used in the L64364.

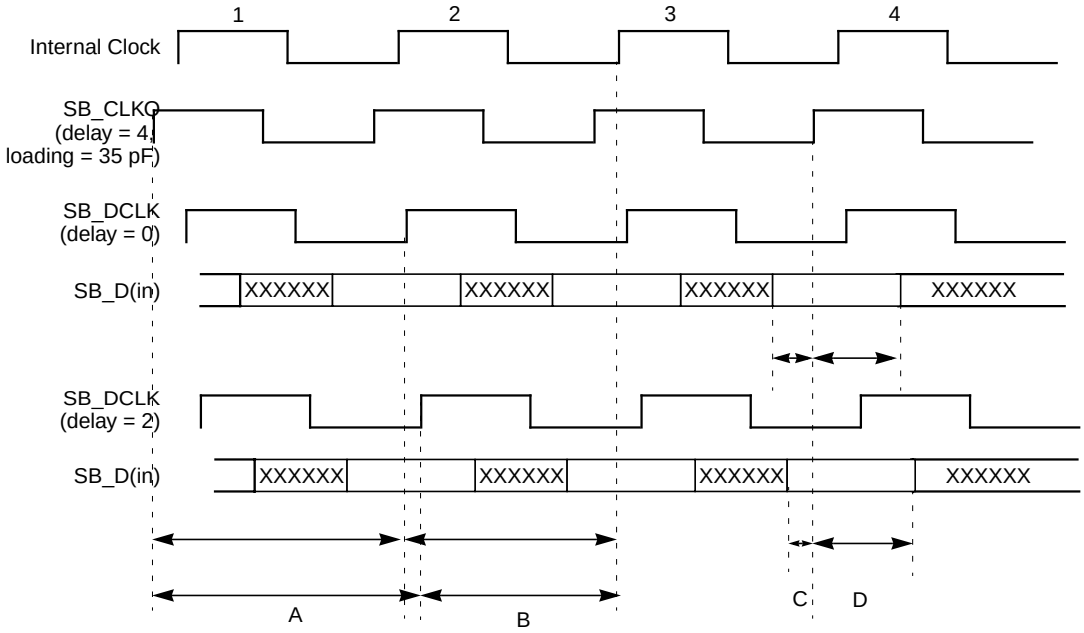
SB_DCLK delay[2:0] **[10:8]**
These bits specify a delay of SB_DCLK from the internal system clock of up to 4 ns in approximate 0.5 ns steps. This delay can be adjusted so that the SB_D[31:0] inputs from external RAM meet the specified ATMizer II+ setup time. The following table shows the delay between

SB_DCLK and the internal system clock for each delay setting.

SB_DCLK delay[2:0]	Delay from Internal Clock (ns)
0b000	+0.4
0b001	+1.1
0b010	+1.5
0b011	+2.2
0b100	+2.7
0b101	+3.4
0b110	+3.8
0b111	+4.5

Should your system design have trouble making the SB_D[31:0] inputs setup time specified in Table 13.2, the value of SB_DCLK delay could be increased to delay the clock that captures the SB_D[31:0] data input. Doing this effectively decreases SB_D[31:0] input setup requirements and increases SB_D[31:0] input hold requirements as shown in Figure 10.4.

Figure 10.4 Effects of SB_DCLK Delay Register



A: Effective clock period from external devices to ATMizer II+ (cycle time – SB_CLK0_delay + SB_DCLK_delay)
B: Effective internal clock period from SB_D register to internal masters (cycle time – SB_DCLK_delay)
C: Effective SB input setup = Input setup time + SB_CLKO_delay – SB_DCLK_delay
D: Effective SB input hold = Input hold time – SB_CLKO_delay + SB_DCLK_delay

R **Reserved** **[7:4]**

Not used in the L64364.

SB_CLKO delay[2:0] **[2:0]**

These bits specify a timing adjustment of SB_CLK0 with respect to the internal system clock of up to ± 2 ns in approximate 0.5 ns steps. The adjustment is provided to compensate for different output loadings on the Secondary Bus. The following table shows the skew between the two clocks based on SB_CLK0 loading and the delay setting. The grayed-out cells in the table show where the L64364 meets its Secondary Bus timing specification.

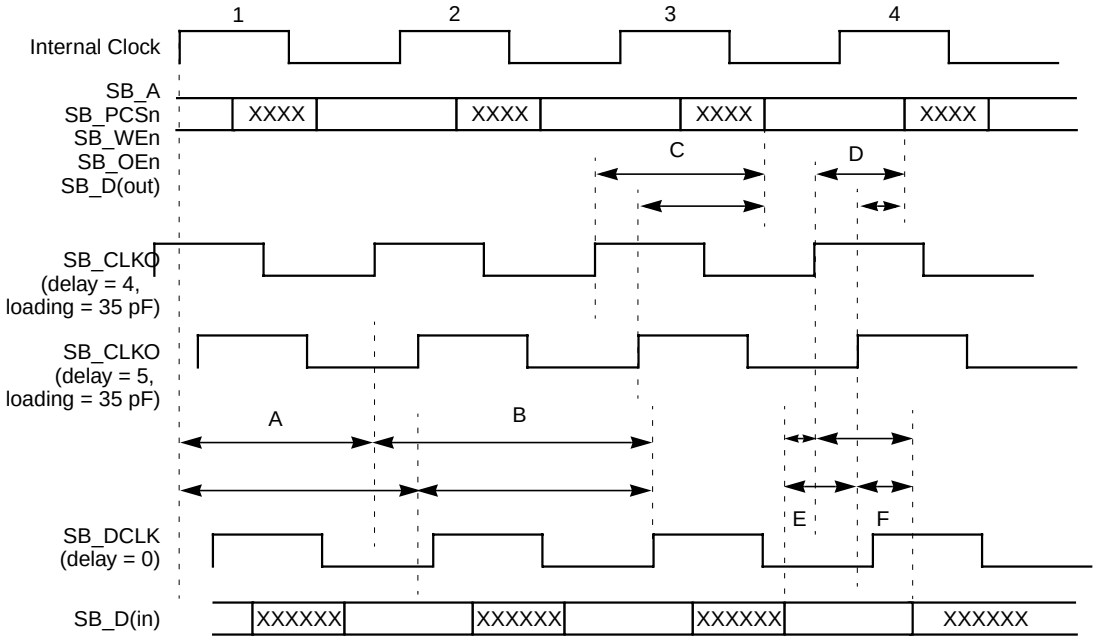
SB_CLK delay[2:0]	SB_CLKO Skew under Loading (ns)[1]			
	15 pF	35 pF	65 pF	85 pF
0b000	–3.1	–2.7	–2.0	–1.6
0b001	–2.7	–2.3	–1.6	–1.1
0b010	–1.9	–1.5	–0.8	–0.3
0b011	–1.5	–1.0	–0.3	+0.1
0b100	–0.6	–0.2	+0.5	+0.9
0b101	-0.2	+0.3	+0.9	+1.4
0b110	+0.5	+1.0	+1.6	+2.1
0b111	+1.0	+1.4	+2.1	+2.5

1. These are preliminary clock skew numbers.

The SB_CLKO clock output is a derivative of the internal system clock and, for proper Secondary Bus operation with synchronous memories, the skew between SB_CLKO and the internal system clock needs to be minimized to prevent setup and hold violations. The SB_CLKO delay control aligns the SB_CLKO clock output to the internal system clock over the 15 pF to 85 pF loading range that is specified for Secondary Bus pins. The variation in loading from 15 pF to 85 pF causes the timing of the SB_CLKO output buffer to change by approximately 2 ns.

The Secondary Bus timing specifications in Table 13.2 are based on the skew between the internal system clock and SB_CLKO being within ± 0.5 ns. However, you may operate the SB interface outside this range if needed by your system's design. Figure 10.5 shows the effects of the delay settings. Increasing clock skew to the positive side effectively decreases output delay, output hold, and input hold times, and increases input setup times. Increasing clock skew to the negative side effectively increases output delay, output hold, and input hold times, and decreases input setup times. The default setting of 4 keeps the skew between SB_CLKO and the internal system clock within ± 0.5 ns for 15 pF to 65 pF SB_CLKO loading.

Figure 10.5 Effects of SB_CLKO Delay Register



A: Effective clock period from ATMizer II+ to external devices (cycle time + SB_CLKO_delay)
B: Effective clock period from external devices to ATMizer II+ (cycle time – SB_CLKO_delay + SB_DCLK_delay)
C: Effective SB output delays = Output valid time – SB_CLKO_delay
D: Effective SB output hold = Output hold time – SB_CLKO_delay
E: Effective SB input setup = Input setup time + SB_CLKO_delay – SB_DCLK_delay
F: Effective SB input hold = Input hold time – SB_CLKO_delay + SB_DCLK_delay

10.3 Secondary Bus Performance Considerations

L64364 throughput directly relates to Secondary Bus usage. For most applications, the L64364 provides 150 Mbytes/s, full-duplex, PDU throughput when operating at 80 MHz with synchronous SRAM.

The performance over the bus is primarily influenced by latencies in the memory access pipeline. From a requesting master's perspective, reads experience latencies in every stage of the pipeline while writes experience latencies only up to the bus arbitration stage. Latencies can be calculated with the following generic formulas:

Total Read Latency = $L_{RM} + L_A + L_{MC} + L_{MD} + L_{RD}$

where:

L_{RM} Requesting Master Latency. This is 3 for the APU. It takes 3 clock cycles to get the memory request from the CW4011 pipeline to the arbiter.

L_A Arbitration Latency. When the requesting master is the default master (last one making a request) and no other requests are active, this latency is at its minimum of 2 clock cycles for the APU and 1 clock cycle for other bus masters. Otherwise, it depends on how long it takes to process higher-priority requests.

L_{MC} Memory Controller Latency. This number is 1. There is a 1 clock-cycle delay from the arbiter to the registers that drive the Secondary Bus pins.

L_{MD} Memory Device Latency. This latency depends on the type of memory device being accessed. Consult your memory databook for this number.

L_{RD} Read Data Latency. It takes 1 clock cycle to get data from the Secondary Bus pins to the requesting master.

Total Write Latency = $L_{RM} + L_A + L_{MD}$*

*Writes only experience memory device latency if extra cycles are needed for address overhead in devices such as DRAM and SDRAM.

Using the above formulas for an APU read from or write to secondary memory yields a minimum of:

Total Read Latency = 3 + 2 + 1 + 3 + 1 = 10 clock cycles

Total Write Latency = 3 + 2 + 0 = 5 clock cycles

In a busy system, latency numbers are much higher because each master must wait for its arbitration slot. During these times, L_{RM}, L_A, L_{MC}, and possibly some L_{MD} are in the current master's access and the generic equation becomes:

Current Master's Total Latency + $L_{MD} + L_{RD}$

In the busy system scenario, memory accesses are packed as tight as the memory devices allow at the Secondary Bus pins. The references in Table 10.3 and Table 10.4 are with respect to memory device timings at the Secondary Bus pins. The tables show just how tightly the L64364 can pack requests to different types of memory.

Table 10.3 lists the minimum number of clocks per data word required during a Secondary Bus burst transfer.

Table 10.3 SBC Clocks per Data Word

Secondary Page	Example Part #	Min Clocks/Data
SDRAM	NEC uPD4516161G5-A10	1
SSRAM	Micron MT58LC32K32C4-7	1
SRAM	Micron MT5LC2568-15	2
PHY		3
EPROM		3

The transfer sequence dictates how many additional lead-off cycles are required for the first word of a burst transfer. Table 10.4 specifies the number of lead cycles needed for different transfer sequences.

Table 10.4 SBC Transfer Lead-Off Cycles

Preceding Transfer		Number of Lead-Off Cycles for Next Transfer					
		SSRAM		SDRAM[2]		ASRAM	
		R	W	R	W	R	W
SSRAM[1]	R	0	1	5	3	1	1
	W	2	0	4	2	1	1
SDRAM[2]	R	3	1	4	2	4	1
	W	5	3	6	4	3	3
ASRAM	R	3	1	4	3	0	1
	W	3	1	6	3	1	0

1. For D7 devices; add one cycle for C4 SSRAM to non-SSRAM transfers.
2. Assumes RAS; CAS latency = 2

10.4 SDRAM Controller

The Synchronous DRAM (SDRAM) Controller connects to 16 Mbit and 64 Mbit SDRAMs from NEC. Configurations of two 1 M x 16-bit, four 2 M x 8-bit, and four 8 M x 8-bit SDRAMS provide 4 Mbytes, 8 Mbytes, or 32 Mbytes of local memory, respectively. Interface signal timing meets the requirements of NEC's 1 M x 16-bit (uPD4516161G5-A10) SDRAMs and 2 M x 8-bit (uPD4516821G5-A10) SDRAMs and 8 M x 8-bit (uPD4564841G5-A10) SDRAMs at a 66 MHz clock rate. Different speed grades may be used in lower performance applications.

Four Micron 8-bit DRAMs (MT41LC256K32D4) may also be used for 1 Mbyte configurations.

All SDRAM interface signals are synchronized to the rising edge of the output clock signal, SB_CLK0. SB_CLK0 has the same frequency as the L64364 system clock (in the recommended configuration).

10.4.1 SDRAM Connections

Table 10.5 lists the pin/signal interconnects between the L64364 and SDRAM devices for 1 Mbyte, 4 Mbyte, and 8 Mbyte configurations.

SDRAM row and column addresses are multiplexed onto Secondary Bus address lines SB_A[15:2]. The SDRAM control signals (RASn and CASn) connect to Secondary Bus address lines SB_A21 and SB_A20, respectively. The SDRAM write enable control signal (WEn) connects to the SB_WEn[0] output. The byte enables for SDRAM write operations are on the output enable pins, SB_OEn[3:0]. For the SDRAM page, SB_OEn[3:0] are active low signals, which is consistent with the SDRAM DQM signal definition.

Table 10.5 ATMizer II+ Chip to SDRAM Interconnections

L64364 Pin	8-Bit Wide Configurations				16-Bit Wide Configurations		32-Bit Wide Configuration
	2 M x 8 8 M x 8 Chip #1	2 M x 8 8 M x 8 Chip #2	2 M x 8 8 M x 8 Chip #3	2 M x 8 8 M x 8 Chip #4	1 M x 16 4 M x 16 8 M x 16 Chip #1	1 M x 16 4 M x 16 8 M x 16 Chip #2	256 K x 32
SB_A[15][1]	A[13]	A[13]	A[13]	A[13]	A[13]	A[13]	–
SB_A[14][1]	A[12]	A[12]	A[12]	A[12]	A[12]	A[12]	–
SB_A[13]	A[11]	A[11]	A[11]	A[11]	A[11]	A[11]	BA
SB_A[12]	A[10]	A[10]	A[10]	A[10]	A[10]	A[10]	A[8]
SB_A[11]	A[9]	A[9]	A[9]	A[9]	A[9]	A[9]	–
SB_A[10]	A[8]	A[8]	A[8]	A[8]	A[8]	A[8]	–
SB_A[9:2]	A[7:0]	A[7:0]	A[7:0]	A[7:0]	A[7:0]	A[7:0]	A[7:0]
SB_D[31:24]	D[7:0]	–	–	–	D[15:8]	–	D[31:24]
SB_D[23:16]	–	D[7:0]	–	–	D[7:0]	–	D[23:16]
SB_D[15:8]	–	–	D[7:0]	–	–	D[15:8]	D[15:8]
SB_D[7:0]	–	–	–	D[7:0]	–	D[7:0]	D[7:0]
SB_A[21]	RASn	RASn	RASn	RASn	RASn	RASn	RASn
SB_A[20]	CASn	CASn	CASn	CASn	CASn	CASn	CASn
SB_WEn[0]	WEn	WEn	WEn	WEn	WEn	WEn	WEn
SB_OEn[3]	DQM	–	–	–	DQMU	–	DQM3
SB_OEn[2]	–	DQM	–	–	DQML	–	DQM2

Table 10.5 ATMizer II+ Chip to SDRAM Interconnections (Cont.)

L64364 Pin	8-Bit Wide Configurations				16-Bit Wide Configurations		32-Bit Wide Configuration
	2 M x 8 8 M x 8 Chip #1	2 M x 8 8 M x 8 Chip #2	2 M x 8 8 M x 8 Chip #3	2 M x 8 8 M x 8 Chip #4	1 M x 16 4 M x 16 8 M x 16 Chip #1	1 M x 16 4 M x 16 8 M x 16 Chip #2	256 K x 32
SB_OEn[1]	–	–	DQM	–	–	DQMU	DQM1
SB_OEn[0]	–	–	–	DQM	–	DQML	DQM0
SB_CLKO	CLK	CLK	CLK	CLK	CLK	CLK	CLK
VDD	CKE	CKE	CKE	CKE	CKE	CKE	CKE
VSS	–	–	–	–	–	–	DSF
SB_PCSn[4]	CSn	CSn	CSn	CSn	CSn	CSn	CSn

1. SB_A[15:14] are required only for 64 Mbit and 128 Mbit SDRAMs.

10.4.2 SDRAM Controller Configuration

Synchronous DRAMs have numerous programmable configuration options. SDRAM configuration settings depend on the L64364's clock frequency and the SDRAM speed grades. The default setting (reset state) of the SP_SDRAM register (Figure 10.6) corresponds to NEC's 16 Mbit SDRAM with -A10 speed grade. The register is at APU address 0xB800.0804.

Figure 10.6 SP_SDRAM Register

31	30	29	28	27 22	21 20	19 18	17 16	15 12	11	10 8	7	6 4	3 2	1	0
R	PC	MRS	REF	R	CL[1:0]	R	RCD[1:0]	RC[3:0]	R	RAS[2:0]	R	DAL[2:0]	R	RP3	DPL2
Default Value & Read/Write Status															
0x0				0x00	0x2	0x0	0x2	0x7	0x0	0x5	0x0	0x3	0x0	0x0	0x0
R	R/W			R	R/W	R	R/W		R	R/W	R	R/W	R	R/W	

R **Reserved** **31**

Not used in the L64364.

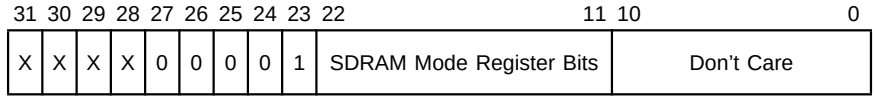
PC **Precharge Command** **30**

PC is the manual Precharge command. When set, it causes the SBC to generate one precharge cycle for both SDRAM banks. This bit is automatically cleared on completion of the Precharge command.

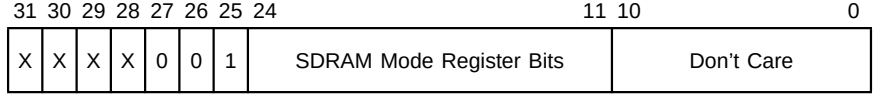
MRS **Mode Register Set** **29**

When MRS is set, the subsequent store word operation to the SDRAM page generates a Mode Register Set command (see Section 10.4.3, “SDRAM Initialization.”) The SDRAM Mode register bits are loaded from the row address bits.

For the 16 Mbyte memory map, typical virtual memory store word addresses are 0xA080.8000, 0xA081.0000, and 0xA081.8000 for CAS latency of 1, 2, or 3 respectively. The format of the Mode Register Set command is:



For the 64 Mbyte memory map, typical virtual memory store word addresses are 0xA200.8000, 0xA201.0000, and 0xA201.8000 for CAS latency of 1, 2, or 3 respectively. The format for the Mode Register Set command is:



The store word data is not used. The SBC clears the MRS bit when the mode register write operation is completed.

REF **Refresh Cycle Command** **28**

The REF bit is a manual Refresh command. When set, this bit causes the SBC to generate one refresh cycle for both SDRAM banks, independent of the state of the refresh timer.

R **Reserved** **[27:22]**

Not used in the L64364.

CL[1:0] **CAS Latency** **[21:20]**
CL[1:0] specifies CAS latency (1 to 3 cycles). The default setting is 2 cycles, which corresponds to 66 MHz operation with NEC SDRAMs with a speed grade of 10.

R **Reserved** **[19:18]**
Not used in the L64364.

RCD[1:0] **Row Activate to Read/Write Command Delay** **[17:16]**
RCD[1:0] selects the number of cycles between a Row Activate command (RAS) and a Read/Write command. If RAS occurs in clock cycle n, the Read/Write command will occur in cycle n + RCD[1:0]. Valid selections are 1, 2, or 3. The default setting is 2.

RC[3:0] **Refresh to Next Command Delay/ Row Activate to Row Activate Delay** **[15:12]**
RC[3:0] selects the minimum number of cycles between the Column Before Row (CBR) Refresh command (REF) and the next command. That is, if REF occurs in cycle n, then the earliest the next command can occur is cycle n + RC[3:0]. RC[3:0] is also the minimum number of cycles between Row Activate commands. Valid settings are 2 through 15. The default setting is 7.

R **Reserved** **11**
Not used in the L64364.

RAS[2:0] **Row Activate to Precharge Command Delay** **[10:8]**
RAS[2:0] specifies the number of clock cycles between the row activate and Precharge commands. That is, if RAS occurs in clock cycle n, Precharge (PRE) will not occur before clock cycle n + RAS[2:0]. Valid settings are 3 through 7. The default setting of 5 is required for NEC SDRAMs with 10 speed grade at 66 MHz.

R **Reserved** **7**
Not used in the L64364.

DAL[2:0] **Write Complete to Refresh/Row Active Command Delay** **[6:4]**
DAL[2:0] specify the minimum number of cycles from the end of a write to the next row activate or Refresh command. That is, if a write transfer completes in cycle n, the earliest the next RAS or Refresh command will occur

is cycle n + DAL[2:0]. Valid settings are 3 through 5. The default value of 3 is required for NEC SDRAMs with 10 speed grade at 66 MHz.

R **Reserved** **[3:2]**
Not used in the L64364.

RP3 **Precharge to Row Activate Command Delay** **1**
When set, RP3 specifies 3 as the minimum number of clocks between a Precharge command and the next Row Activate command. When cleared, RP3 specifies 2 as the minimum number of clocks. Cleared is the default setting for this bit.

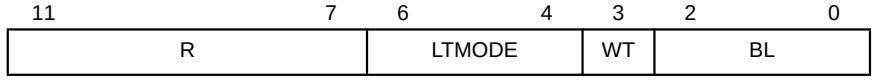
DPL2 **Write Complete to Precharge Command Delay** **0**
When set, DPL2 causes a one cycle delay between the last word of a write and the Precharge command. When DPL2 is cleared (the default setting), the write is terminated by a Precharge command in the cycle following the last valid write data cycle.

10.4.3 SDRAM Initialization

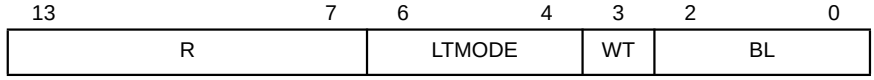
SDRAMs must be initialized at power up. Precharge must be the first command to the SDRAM. Following the Precharge command, a Mode Register Set command must be issued to configure the SDRAM. When set, the MRS bit in the SP_SDRAM register generates the Mode Register Set command (refer to page 10-17).

The format of the 16 Mbit SDRAM Mode register is shown in Figure 10.7.

Figure 10.7 SDRAM Mode Register



The format of the 64 Mbit SDRAM Mode register is:



R **Reserved** **[13:7]**
Not used in the L64364. Must be kept set at 0.

LTMODE **CAS Latency** **[6:4]**
LTMODE selects the CAS latency and must be set to the same value programmed in the CL[1:0] bits in the SDRAM Control register.

WT **Burst Sequence** **3**
WT selects a sequential or interleaved burst sequence. Since the L64364's burst length is 1, the value of this bit is *don't care*.

BL **Burst Length** **[2:0]**
BL specifies the burst length. It must be set to 0b000 for a burst length of 1 since a Read/Write command is issued with each CAS.

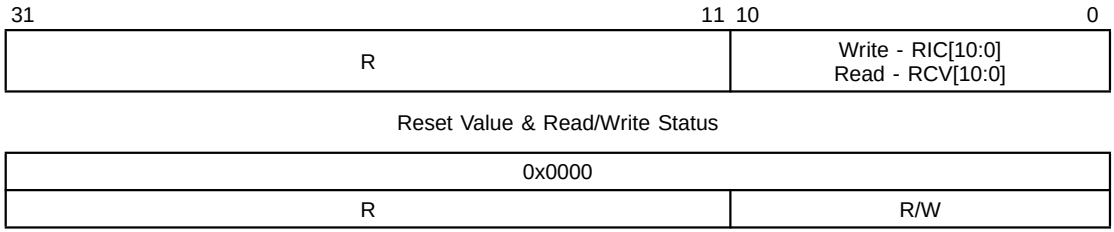
After programming the SDRAM Mode register, two CBR refresh cycles must transpire before the SDRAM becomes accessible. CBR refresh can be initiated using either the Refresh Interval Timer (see the next topic) or by setting the REF bit (page 10-17) in the SDRAM Control register.

10.4.4 SDRAM Refresh

16 Mbit SDRAMs require 2,048 row-refresh cycles in 32 ms and 64 Mbit SDRAMs require 4,096 row-refresh cycles in 64 ms, which creates a maximum refresh interval of 15,625 ns. If the L64364 clock frequency is 80 MHz, a Refresh command must be issued every 1,250 clock cycles and, if the clock frequency is 100 MHz, a Refresh command must be issued every 1,562 clock cycles. To perform this function, the SDRAM Controller includes an 11-bit Refresh Interval Timer.

The Refresh Interval Timer has an 11-bit interval time initial register and an 11-bit counter which is decremented each clock cycle. The SP_Refresh register is accessible by the APU at physical memory address 0xB800.0808. The format for the SP_Refresh register is shown in Figure 10.8.

Figure 10.8 SP_Refresh Register



R **Reserved** **[31:11]**
Not used in the L64364.

RIC[10:0] **Refresh Interval Count** **Write [10:0]**
RIC[10:0] selects the number of L64364 system clock cycles between refresh commands. For 80 MHz operation, RIC[10:0] should be programmed to a value no greater than 0x480 (1152) and for 100 MHz operation, RIC[10:0] shoudl be programmed to a value no greater than 0x61A (1562) to ensure refresh occurs within the 32/64 ms time limit. RIC[10:0] = 0x000 disables the SDRAM refresh logic. RIC[10:0] is write only.

RCV[10:0] **Refresh Counter Value** **Read [10:0]**
RVC[10:0] is the state of the Refresh Counter. The Secondary Bus Controller issues a CBR Refresh command when the Refresh Counter wraps. RVC[10:0] is read only.

Following PCI_RSTn, the Refresh Interval Count and the Refresh Counter both equal 0. Writing a nonzero value to RIC[10:0] activates the Refresh Interval Timer. Starting from the same initial value as RIC[10:0], the Refresh Counter decrements each L64364 system clock cycle. When the Refresh Counter reaches zero, the Secondary Bus Memory Controller issues a CBR Refresh command and wraps the count to the initial RIC[10:0] value.

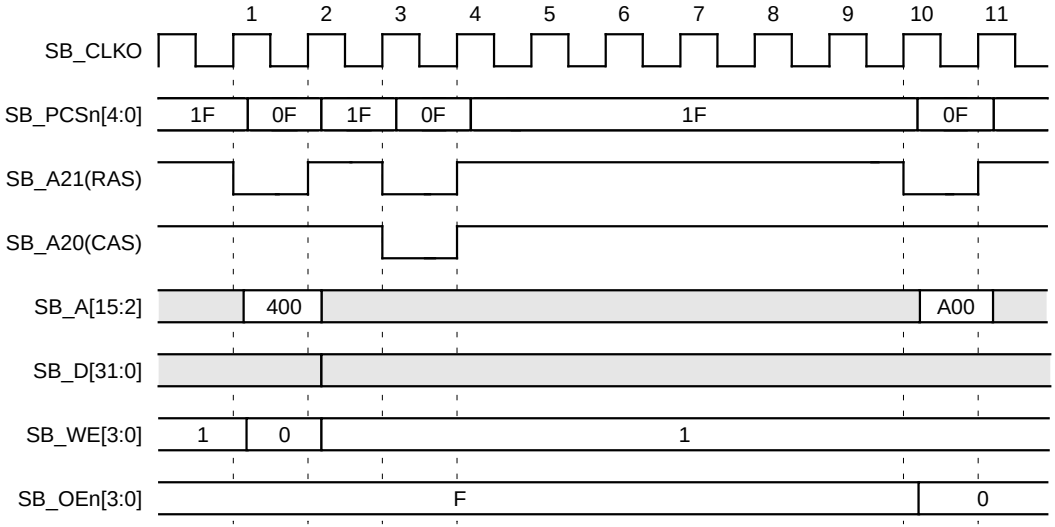
If a data transaction is occurring when the count reaches zero, the Secondary Bus Controller waits for completion of the data transaction and then issues the Refresh command.

Figure 10.9 illustrates Refresh command timing. In the figure, a Precharge command is issued on clock edge 2, either from a previous read/write transfer or from the manual Precharge command.

The timing from precharge to refresh is selected by the RP3 bit in the SP_SDRAM register (page 10-16). In Figure 10.9, RP3 is set to 0, causing the CBR Refresh command to occur two cycles later on clock edge 4.

The SP_SDRAM register's RC[3:0] bits specify the earliest the next Row Activate (RAS) command can occur. In this example, an activate command occurs on clock edge 10, since RC is set to 7.

Figure 10.9 SDRAM Refresh Timing



10.4.5 Secondary Bus Time-Out

The SDRAM refresh sequence does not need to be reinitiated following a Secondary Bus time-out.

10.4.6 SDRAM Command Summary

This section summarizes the commands generated by the SDRAM Controller. Table 10.6 lists the SDRAM signal values for each SDRAM command.

Table 10.6 SDRAM Command Summary

Command	CSn[1]	RASn[1]	CASn[1]	WEn[1]	A[13][2,4]	A[12][2]	A[11][2]	A[10][2]	A[9:0][2]
No Operation	1	X	X	X	X	X	X	X	X
Mode Reg Set	0	0	0	0	AD[24]	AD[23]	AD[22]	AD[21]	AD[20:11]
Row Activate	0	0	1	1	AD[23]	AD[22]	AD[11]	AD[10]	AD[21:12]
Precharge	0	0	1	0	X	X	X	1	X
Write	0	1	0	0	AD[23]	AD[22]	AD[11]	0	AD[23,22,9:2][3] AD[25,24,9:2][4]
Read	0	1	0	1	AD[23]	AD[22]	AD[11]	0	AD[23,22,9:2][3] AD[25,24,9:2][4]
CBR Refresh	0	0	0	1	X	X	X	X	X

1. A “0” in this column indicates asserted, since CSn, RASn, CASn, and WEn are low active.
2. SDRAM pins A[n] = L64364 pins SB_A[n + 2], AD[n] are either L64364 or PCI sourced address bits.
3. Mappings for 16 Mbit SDRAMs only. (SP_Ctrl register bit SB_64M cleared)
4. Mappings for 64 Mbit SDRAMs only. (SP_Ctrl register bit SB_64M set)

The SDRAM Controller deasserts SB_PCSn[4] to idle the SDRAM. The No Operation, Self-Refresh Entry, and Burst Stop commands are not used. The Automatic Precharge mode is also not used.

10.4.7 SDRAM Read Transfer

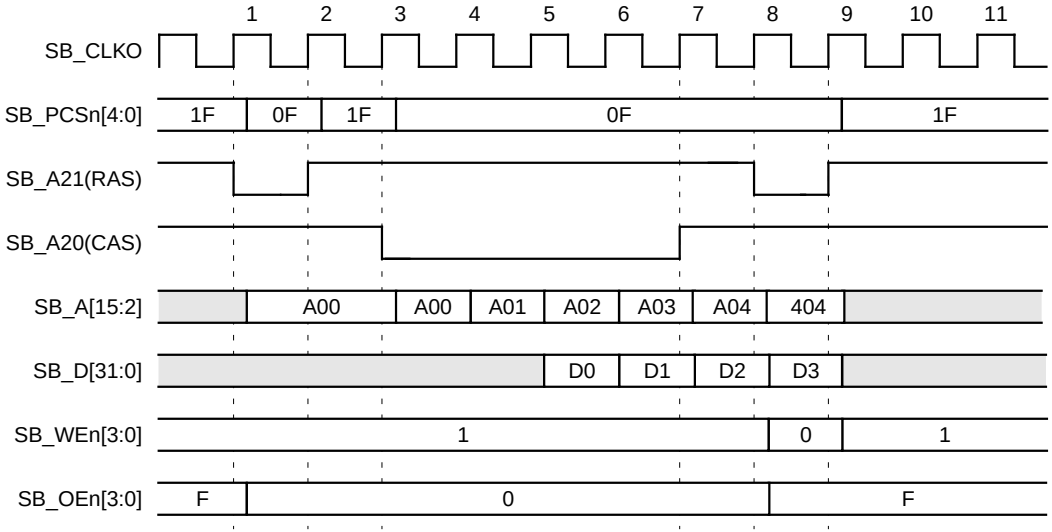
A read transfer is initiated with a Row Activate (RAS) command followed by one or more Read commands and terminated with a Precharge command. The RAS command may be delayed to meet the minimum timing between consecutive RAS commands as specified in the SDRAM Control register’s RC[3:0] field.

On burst reads, a column address is provided for each data cycle. Therefore, the SDRAM burst length (BL) must equal zero and burst type (WT) is a don't care.

Each read transfer is terminated with a Precharge command.

Figure 10.10 depicts timing for an SDRAM Read command. This figure illustrates a four-word, burst read starting from address A00. The SDRAM Control register specifies RAS latency in RCD[1:0] and CAS latency in CL[1:0]. Both fields are set to 2.

Figure 10.10 SDRAM Read Timing



10.4.8 SDRAM Write Transfer

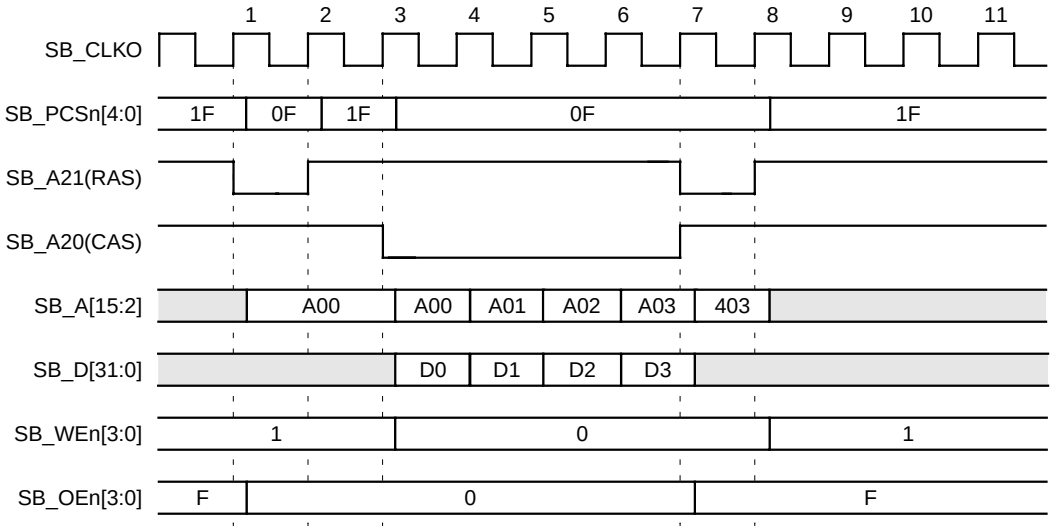
As with read transfers, the write transfer starts with a RAS command followed by one or more Write commands and terminated with a Precharge command. The RAS command may be delayed to meet the minimum timing between consecutive RAS commands as specified in the SDRAM Control register's RC[3:0] field.

On burst writes, a column address is provided for each data cycle. Therefore, the SDRAM burst length (BL) and burst type (WT) are don't care.

The Data Mask (DQM) pins are asserted consistent with the write enables from the L64364 or PCI Bus.

Figure 10.11 depicts timing for an SDRAM Write command. This figure illustrates a four-word, burst write starting from address 0xA00. The SDRAM Control register specifies RAS latency in RCD[1:0] and CAS latency in CL[1:0]. Both fields are set to 2.

Figure 10.11 SDRAM Write Timing



10.5 SSRAM Controller

The Synchronous SRAM Controller directly connects to Synchronous SRAMs from Micron, Motorola, and NEC. Memory configurations of 128 Kbytes to 1 Mbyte are supported. Table 10.7 lists the recommended SSRAM configurations. SSRAM Controller timing is designed to interoperate with pipelined SSRAMs.

Table 10.7 SSRAM Configurations

SSRAM Configuration	SSRAM Part No.	No. of Devices	Device Size
128 Kbytes	- Micron MT58LC32K32D7 - Micron MT58LC32K32C4 - Motorola MCM69P532 - NEC uPD431232LGF	1 1 1 1	32 K x 32 32 K x 32 32 K x 32 32 K x 32
256 Kbytes	- Micron MT58LC64K32D7 - Micron MT58LC64K32C4 - Motorola MCM69P618	1 1 2	64 K x 32 64 K x 32 64 K x 18
512 Kbytes	- Micron MT58LC128K32D7 - Micron MT58LC128K32C4	1 1	128 K x 32 128 K x 32
1 Mbyte	- Micron MT58LC128K32D7 - Micron MT58LC128K32C4	2[1] 2[1]	128 K x 32 128 K x 32

1. In this configuration, address line SB_A[19] is used to select the SSRAM bank.

L64364 to SSRAM interconnect is summarized in Table 10.8.

Table 10.8 SSRAM Interconnections

L64364 Pin	SSRAM Pin
SB_D[31:0]	DQ[32:1]
SB_A[n + 3][1] SB_A[n + 2:2]	CE2, CE2n A[n:0]
SB_PCSn[3]	CEn
SB_OEn[0]	OEn
SB_WEn[3:0]	BW[4:1]
SB_CLKO	CLK
VDD (3.3 V)	GWn, ADSPn, ADVn
VSS	BWEn, MODE, ZZ, ADSCn

1. When designing with multiple banks of SSRAMs, only SB_A[21:15] may be used for secondary chip enables CE2 and CE2n.

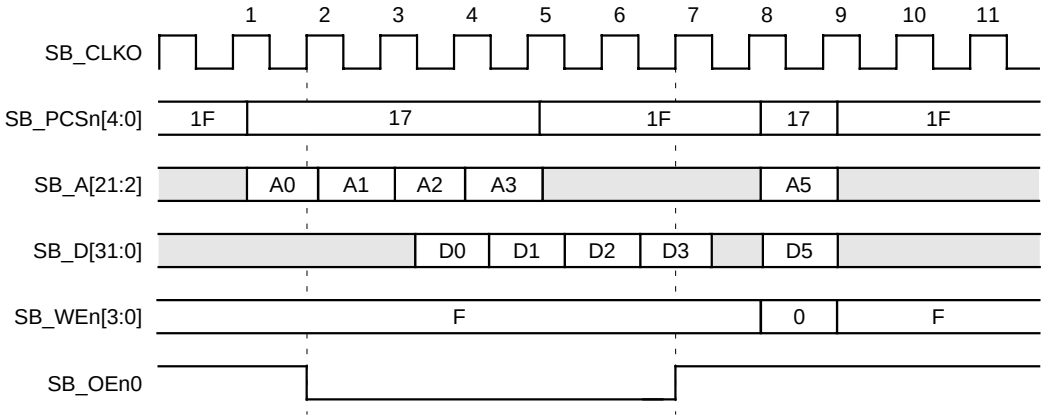
Unlike the SDRAM, there are no programmable options for the SSRAM.

10.5.1 SSRAM Read Transfers

Figure 10.12 illustrates SSRAM read timing. In this example, a four-word burst read from address A0 is followed by a word write to address A5. SB_DSL3 is cleared for this example.

SSRAM read operations start with an address cycle, designated by assertion of SB_PCSn[3]. Read data drives SB_D[31:0] during the next clock cycle. During burst transfers, the burst address drives SB_A[21:2] during each cycle. The Burst read terminates when the SSRAM Controller deasserts PCSn[3], unless the next pending transfer is targeted to SSRAM.

Figure 10.12 SSRAM Read Timing

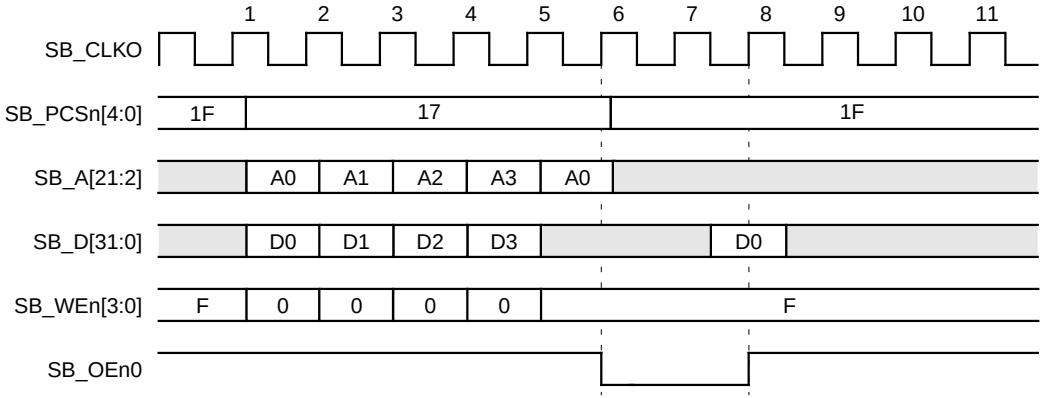


10.5.2 SSRAM Write Transfers

Figure 10.13 shows the SSRAM write timing. In this example, a four-word burst write to address A0 is followed by a word read of address A0. SB_DSL3 is cleared for this example.

SSRAM write transfers start with the assertion of PCSn[3] and one or more write enables, SB_WEn[3:0]. During this cycle, the valid write address is specified on SB_A[21:2], and write data is available on SB_D[31:0]. The burst address drives SB_A[21:2] during each write cycle. A burst write terminates with the deassertion of SB_PCSn[3], unless the next pending transfer is also targeted to SSRAM.

Figure 10.13 SSRAM Write Timing



10.6 32-Bit SRAM/EPROM Controller

For lower performance applications, the 32-bit SRAM/EPROM Controller serves as an alternative to SDRAM or SSRAM. For this memory type, the L64364 directly controls operation of the asynchronous SRAM or EPROM. Configuring the memory is straight forward. The L64364 address, data, and control lines directly connect to corresponding SRAM or EPROM pins. SB_PCSn[2] is used for the chip select. All interface signals synchronize with the SB_CLKO output signal.

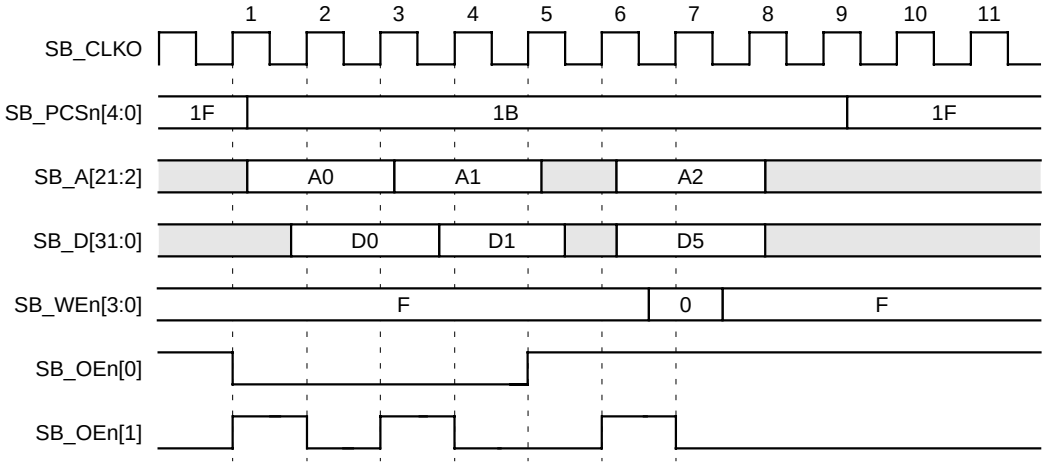
10.6.1 32-Bit SRAM/EPROM Read Transfer

A 32-bit SRAM/EPROM read operation starts with the assertion of SB_PCSn[2]. During this cycle, the read address drives SB_A[21:2]. For the 32-bit EPROM/SRAM page, SB_OEn[1] is used as a high-active address strobe. The address strobe is active when a new address is presented on SB_A[21:2]. Read data is captured in the following cycle if the wait state count SB_Wait2[3:0] = 0x0. Otherwise, read data sampling occurs during the clock cycle specified by SB_Wait2[3:0].

If SB_Wait2[3:0] = 0xF, then SB_RDYn timing is selected and read data is sampled in the clock cycle after resynchronization.

Figure 10.14 shows the SRAM read timing with SB_Wait2 = 0. In this example, a two-word read starts from address A0 followed by a one-word write to address A2.

Figure 10.14 SRAM Read Timing



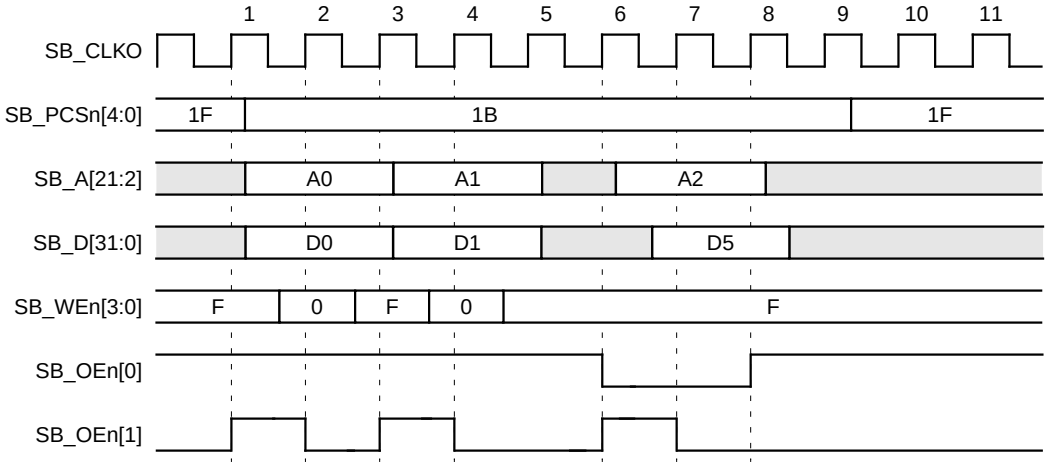
10.6.2 32-Bit SRAM Write Transfers

A 32-bit SRAM write operation starts with the assertion of SB_PCSn[2] on the rising edge of SB_CLKO. During this cycle, SB_A[21:2] contains a valid write address, and SB_D[31:0] contains valid write data. One-half cycle later, on the falling edge of SB_CLKO, the L64364 asserts the write enables, SB_WEn[3:0]. The write enables, address and data remain valid for the number of cycles specified by SB_Wait2[3:0]. If SB_Wait2[3:0] = 0x0, the write enables remain active for one clock cycle (from negative-edge to negative-edge of SB_CLKO).

If SB_Wait2[3:0] = 0xF, then the SB_RDYn input determines when the data is written. See Section 10.6.3, “32-Bit SRAM/EPROM SB_RDYn Timing.”

Figure 10.15 depicts SRAM write transfer timing. In this example, a two-word write starts from address A0. Then, a one-word read from address A2 occurs with SB_Wait2 = 0x0.

Figure 10.15 SRAM Write Timing



10.6.3 32-Bit SRAM/EPROM SB_RDYn Timing

Figure 10.16 and Figure 10.17 show the read and write timing with SB_RDYn. When SB_Wait2 = 0xF, the L64364 uses the asserting edge of SB_RDYn to determine when the transfer is complete. An asserting edge of SB_RDYn must be seen for each word transferred. If asserted synchronously with SB_CLKO, SB_RDYn need only be asserted/deasserted for one clock cycle; otherwise, it must be asserted/deasserted for a minimum of two clock cycles.

For burst transfers, SB_OEn[1] is used as a high-active address strobe indicating that a new address has been placed on SB_A[21:2]. If SB_RDYn is used to complete the transfer, then the logic generating the SB_RDYn pulse should key off of the assertion of SB_OEn[1].

The L64364 captures read data five clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_OEn[0] or, in back-to-back accesses, at the asserting edge of SB_OEn[1] (at the change of address). The L64364 completes write operations four and one-half clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_WEn[3:0].

Figure 10.16 32-Bit SRAM/EPROM Read Timing with SB_RDYn

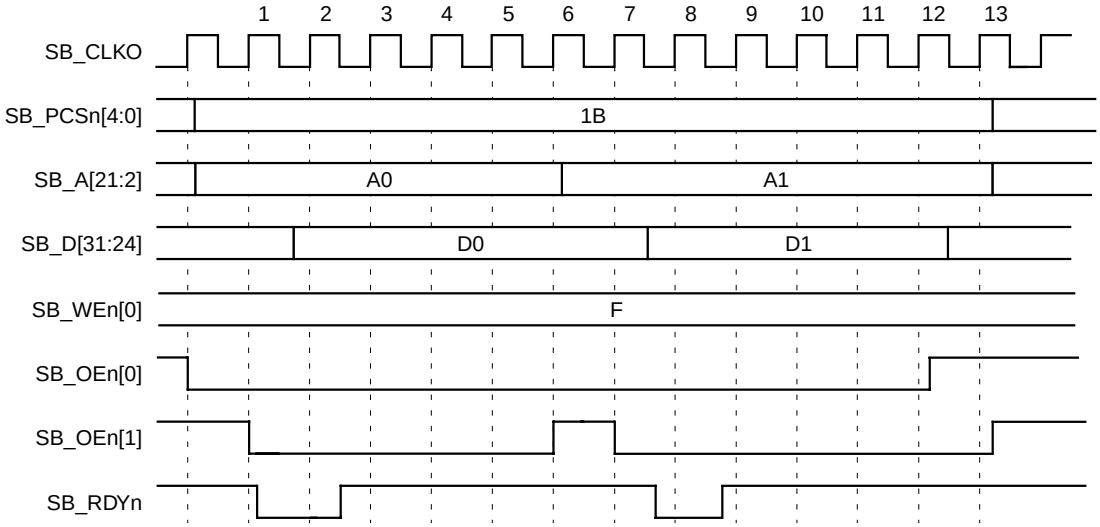
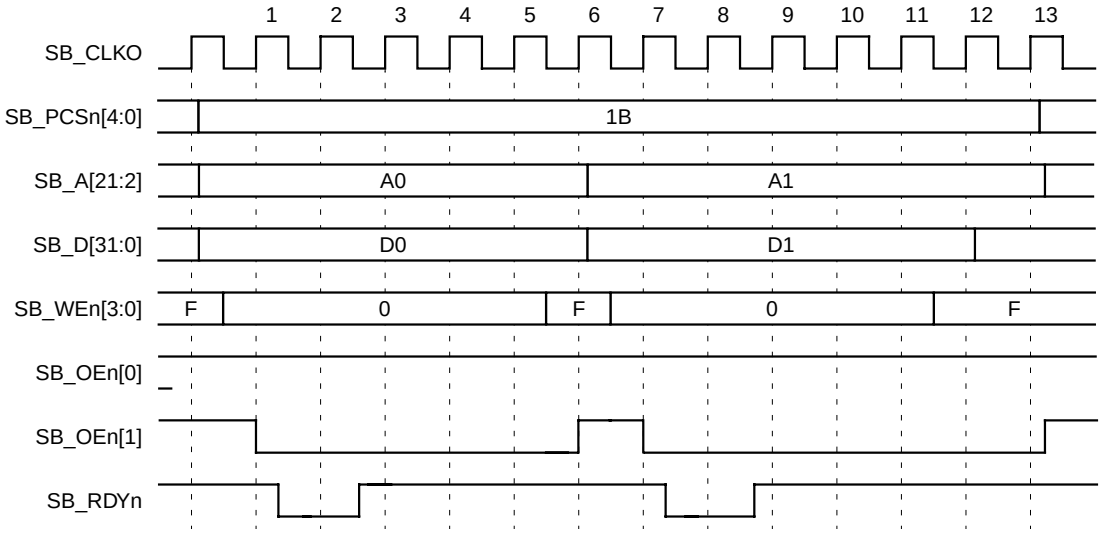


Figure 10.17 32-Bit SRAM/EPROM Write Timing with SB_RDYn



10.7 PHY Controller

The PHY Controller makes it easy to connect the 8-bit control/ management interface common on ATM Physical Layer devices and conforms to the timing specified in Appendix A2.4.2 of the *ATM Forum Utopia Level 2 Specification, Version 1.0*. PHY devices connect to the Secondary Bus as listed in Table 10.9.

Table 10.9 Secondary Bus to PHY Device Connections

Device Connections	Description	Secondary Bus Connections
Addr[19:0]	Contains the Byte Address	SB_A[21:2]
Data[7:0]		SB_D[31:24]
Sel	Chip Select	SB_PCSn[1]
Rd	Read Strobe	SB_OEn[0]
Wr	Write Strobe	SB_WEn[0]
R/W	Read/Write Signal	SB_OEn[2]
DS	Data Strobe	SB_OEn[0] and SB_WEn[0]
Rdy/Dtack	Ready/Data Acknowledge	SB_RDYn
ALE	Address Latch Enable	SB_OEn[1]

10.7.1 PHY Read Transfers

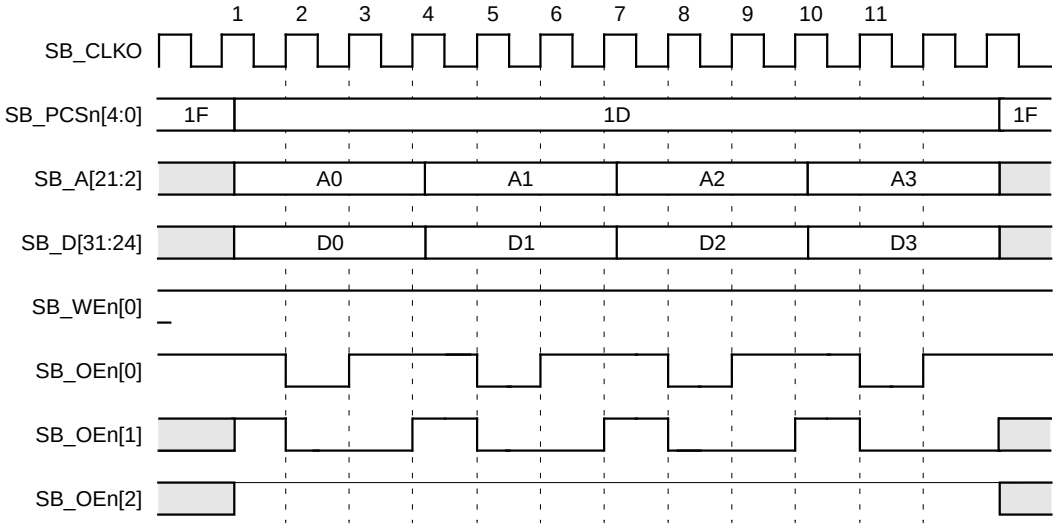
Figure 10.18 illustrates PHY read timing with `SB_Wait1` = 0x0 (zero wait states). Section 10.7.3, "PHY SB_RDYn Timing," describes PHY read/write timing when `SB_Wait1` = 0xF.

PHY read transfers start with the assertion of `SB_PCSn[1]`. During the same cycle, `SB_A[21:2]` has a valid address and the controller asserts an ALE signal on `SB_OEn[1]`. `SB_OEn[2]`, the R/W signal, is also valid at this time.

`SB_Wait1[3:0]` specifies how many cycles the controller asserts `SB_OEn[0]`. If `SB_Wait1[3:0]` equals 0x0 or 0x1, the controller asserts

SB_OEn[0] the cycle after SB_OEn[1] (ALE). Otherwise, SB_OEn[0] lags by two cycles. Read data capture occurs in the last cycle the controller asserts SB_OEn[0].

Figure 10.18 PHY Read Timing



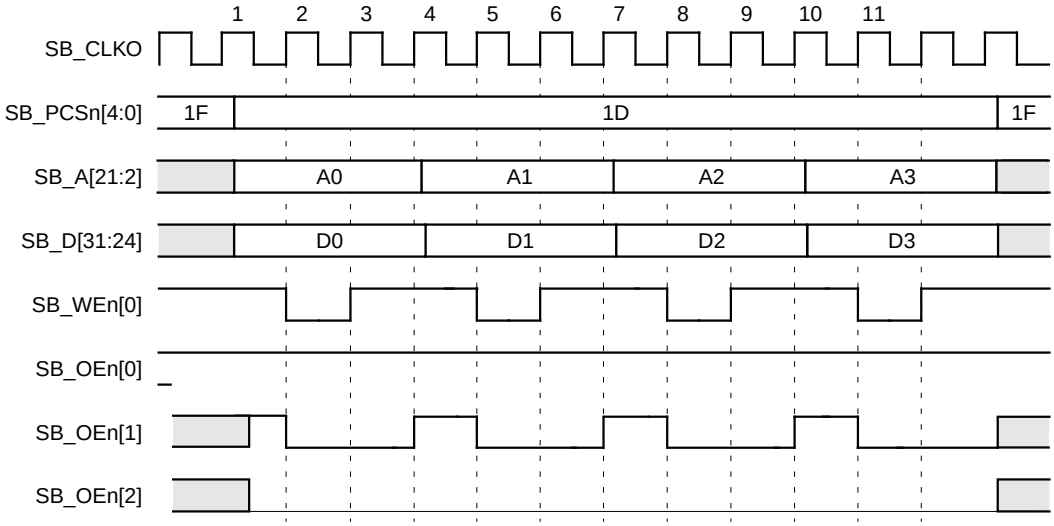
10.7.2 PHY Write Transfers

Figure 10.19 illustrates PHY write transfer timing with SB_Wait1 = 0x0 (zero wait states). Section 10.7.3, "PHY SB_RDYn Timing," describes PHY read/write timing when SB_Wait1 = 0xF.

As with read transfers, PHY write transfers start with the assertion of SB_PCSn[1]. During this same cycle, the controller asserts an ALE signal on SB_OEn[1] and outputs a valid address on SB_A[21:2].

SB_Wait1[3:0] specifies how many cycles the controller asserts SB_WE[0]. If SB_Wait1[3:0] equals 0x0 or 0x1, the controller asserts SB_WEn[0] the cycle after SB_OEn[1] (ALE). Otherwise, SB_WEn[0] lags by two cycles.

Figure 10.19 PHY Write Timing



10.7.3 PHY SB_RDYn Timing

When SB_Wait1 = 0xF, the L64364 uses the asserting edge of SB_RDYn to determine when to complete the transfer. An asserting edge of SB_RDYn must be seen for each byte transferred. Timing for the assertion of SB_RDYn should be based on the asserting edge of SB_OEn[0] for reads (Figure 10.20) and SB_WEn[0] for writes (Figure 10.21). Once asserted, SB_RDYn must remain asserted until the L64364 deasserts SB_OEn[0] for reads and SB_WEn[0] for writes. This conforms to the timing specified in Appendix A2.4.2 of the *ATM Forum Utopia Level 2 Specification, Version 1.0*.

The L64364 captures read data four clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_OEn[0]. It completes write operations four clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_WEn[0].

Figure 10.20 PHY Read Timing with SB_RDYn

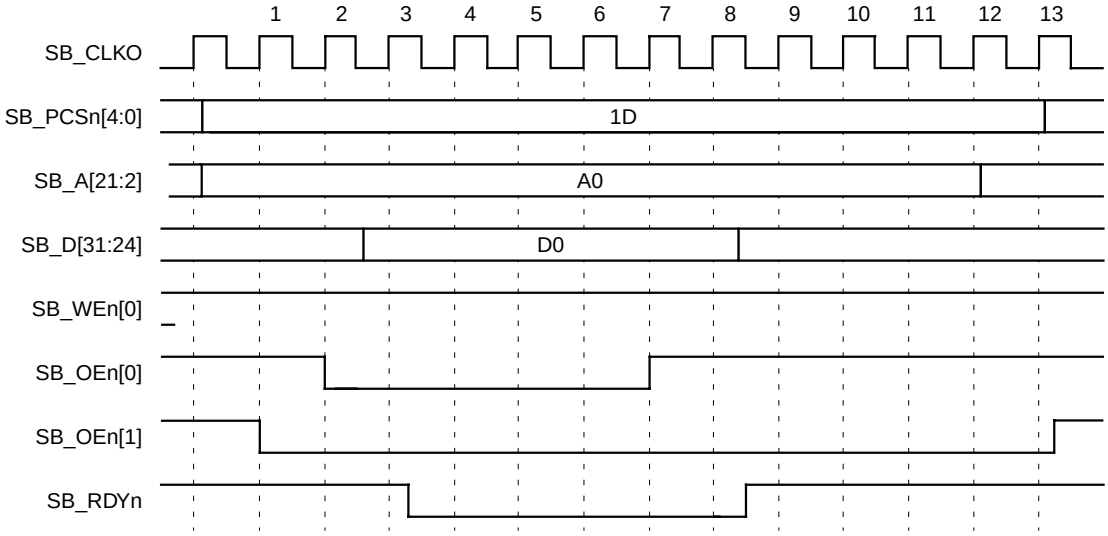
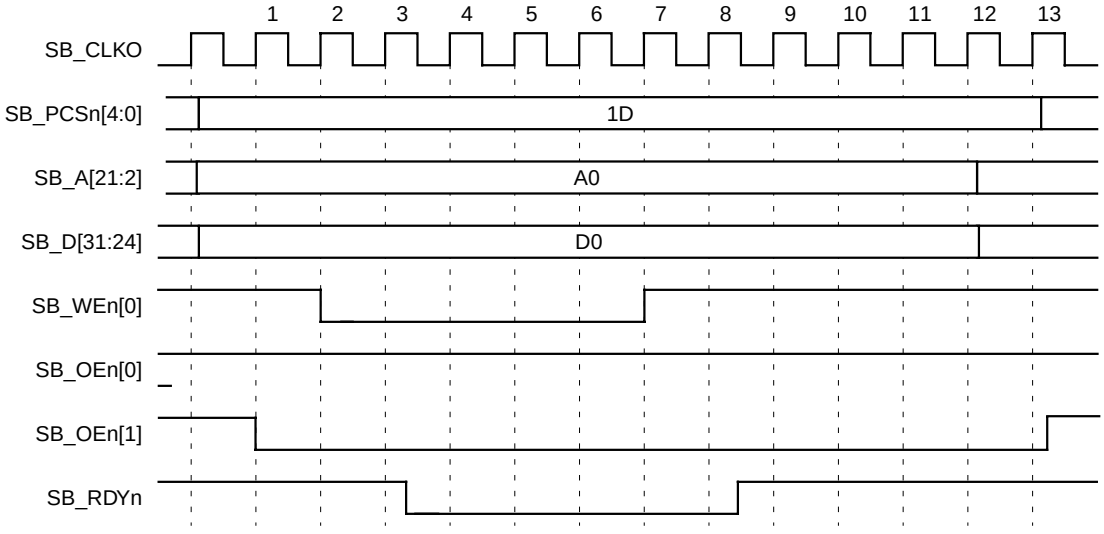


Figure 10.21 PHY Write Timing with SB_RDYn



10.8 8-Bit SRAM/EPROM Controller

The eight-bit SRAM/EPROM Controller supports booting the L64364 from a byte-wide EPROM. Since data transfers are byte wide, the SB_A[21:2] outputs contain the byte address, which are equivalent to the L64364 or PCI address bits [19:0]. Data is transferred on SB_D[31:24], and SB_WEn[0] and SB_OEn[0] are the corresponding write enable and output enable signals.

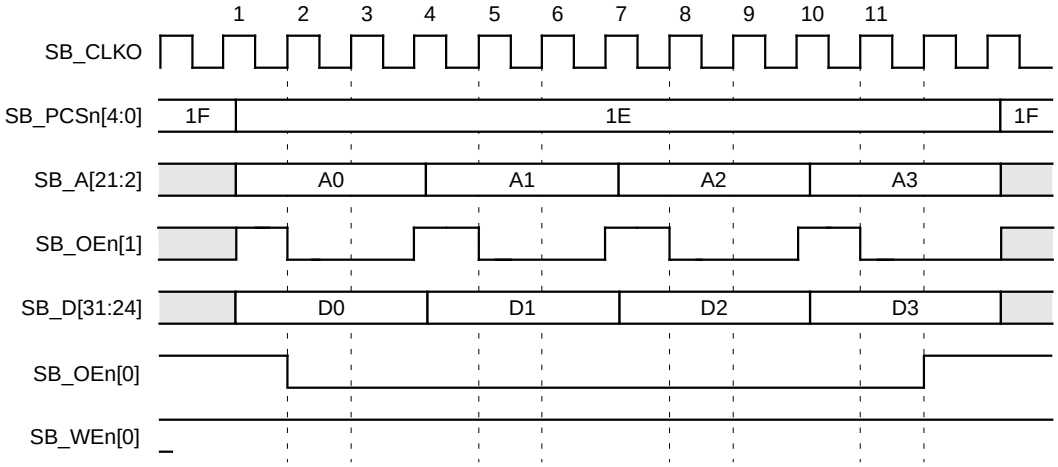
10.8.1 8-Bit SRAM/EPROM Read Transfers

Figure 10.22 shows SRAM/EPROM read timing with SB_Wait1 = 0x0 (zero wait states). Section 10.8.3, "8-Bit SRAM/EPROM SB_RDYn Timing," describes SRAM/EPROM read/write timing when SB_Wait1 = 0xF.

Read transfers start when the controller asserts SB_PCSn[0]. During the same cycle, SB_A[21:2] has a valid address, and the controller asserts an ALE signal on SB_OEn[1].

SB_Wait0[3:0] specifies how many cycles the controller asserts SB_OEn[0]. If SB_Wait1[3:0] equals 0x0 or 0x1, the controller asserts SB_OEn[0] the cycle after SB_OEn[1] (ALE). Otherwise, SB_OEn[0] lags by two cycles. Read data capture occurs in the last cycle the controller asserts SB_OEn[0] or, in the case of bursts, the clock edge on which SB_OEn[1] is asserted (new address).

Figure 10.22 8-Bit SRAM/EPROM Read Timing



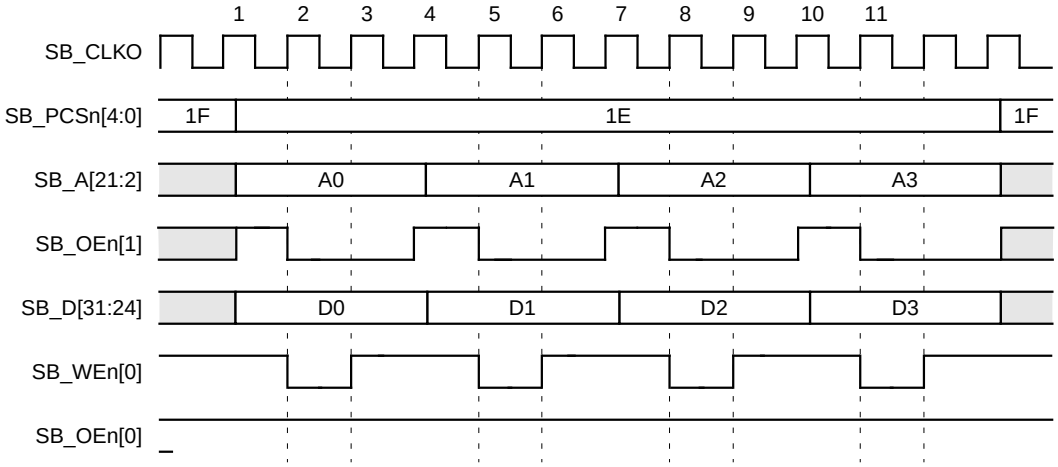
10.8.2 8-Bit SRAM/EPROM Write Transfers

Figure 10.23 illustrates 8-bit write timing with SB_Wait1 = 0x0 (zero wait states). Section 10.8.3, "8-Bit SRAM/EPROM SB_RDYn Timing," describes the SRAM/EPROM read/write timing when SB_Wait1 = 0xF.

As with read transfers, 8-bit SRAM/EPROM write transfers start when the controller asserts SB_PCSn[0]. During this same cycle, the controller asserts an ALE signal on SB_OEn[1] and outputs a valid address on SB_A[21:2].

SB_Wait1[3:0] specifies how many cycles the controller asserts SB_WEn[0]. If SB_Wait1[3:0] equals 0x0 or 0x1, the controller asserts SB_WEn[0] the cycle after SB_OEn[1] (ALE). Otherwise, SB_WEn[0] lags by two cycles.

Figure 10.23 8-Bit SRAM Write Timing



10.8.3 8-Bit SRAM/EPROM SB_RDYn Timing

When SB_Wait1 = 0xF, the L64364 uses the asserting edge of SB_RDYn to determine when the transfer is complete. Refer to Figure 10.24 and Figure 10.25. An asserting edge of SB_RDYn must be seen for each byte transferred. If asserted synchronously with SB_CLKO, SB_RDYn need only be asserted/deasserted for one clock cycle; otherwise, it must be asserted/deasserted for a minimum of two clock cycles.

For burst transfers, SB_OEn[1] is used as a high-active address strobe indicating that a new address has been placed on SB_A[21:2]. If SB_RDYn is used to complete the transfer, then the logic generating the SB_RDYn pulse should key off of the assertion of SB_OEn[1].

The L64364 captures read data five clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_OEn[0] or, in back-to-back accesses, at the asserting edge of SB_OEn[1] (at the change of address). The L64364 completes write operations four clock cycles after the asserting edge of SB_RDYn, at the deasserting edge of SB_WEn[0].

Figure 10.24 8-Bit SRAM/EPROM Read Timing with SB_RDYn

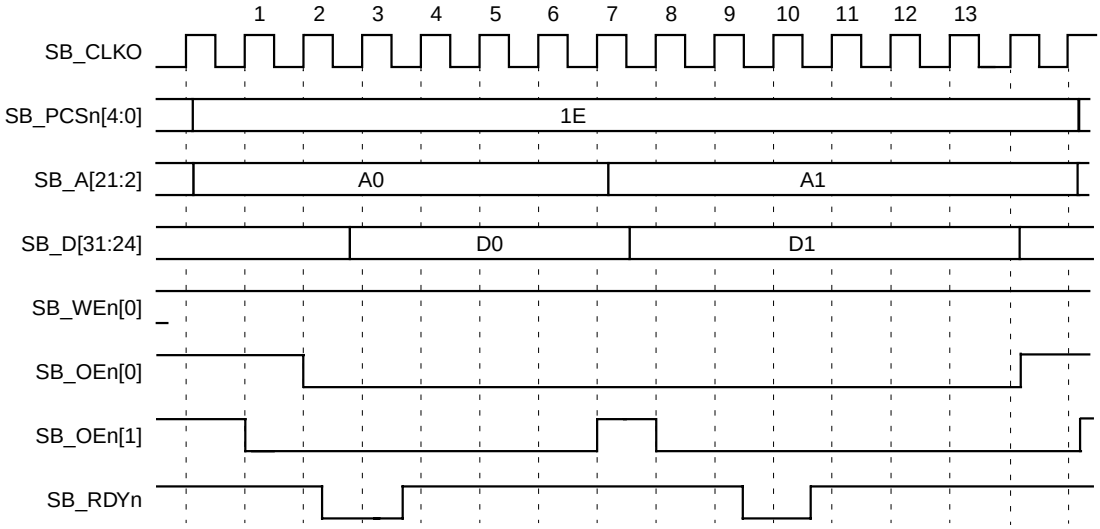
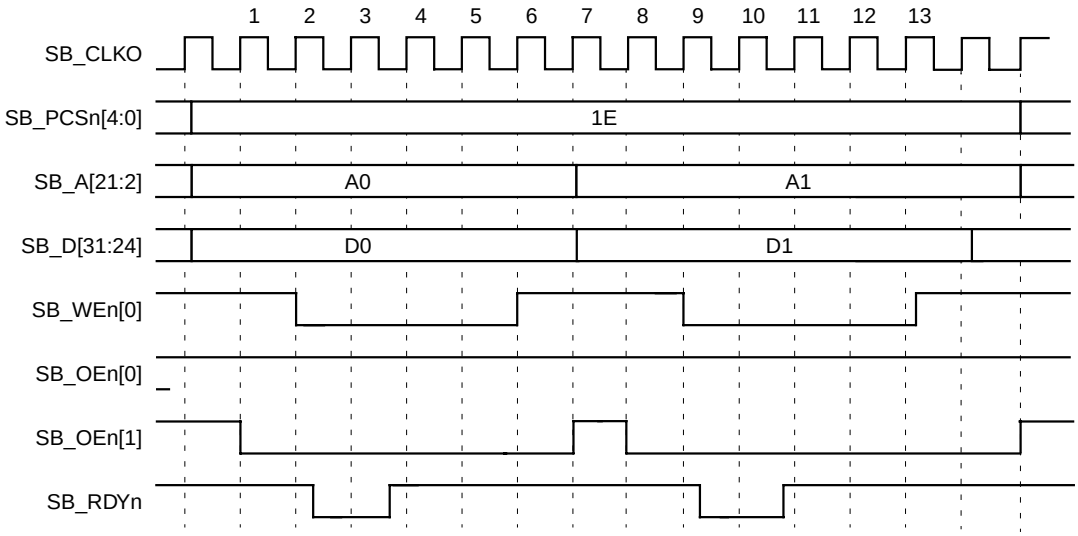


Figure 10.25 8-Bit SRAM/EPROM Write Timing with SB_RDYn



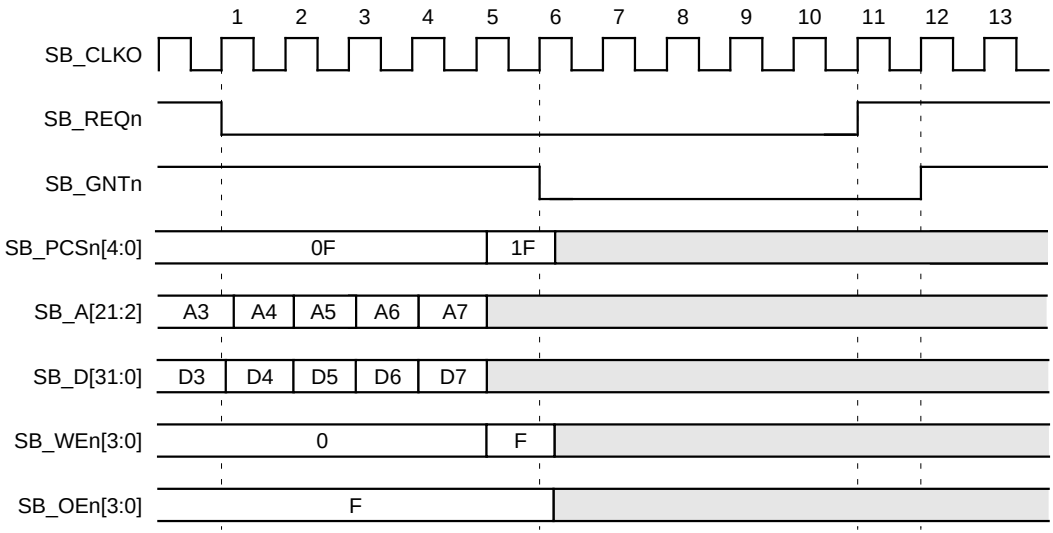
10.9 External Bus Masters

The L64636 can accommodate additional external bus masters on the Secondary Bus. The SB_REQn and SB_GNTn signals allow an external bus master to arbitrate for bus access. Arbitration for the Secondary Bus uses a round-robin technique among the L64364 bus masters (APU, EDMA, and Scheduler) and the PCI Slave interface.

Secondary Bus ownership is granted to the external bus master when SB_GNTn is asserted. Figure 10.26 illustrates Secondary Bus grant timing. All Secondary Bus interface signals, except SB_CLKO and SB_GNTn, are held 3-stated until the external master deasserts SB_REQn. SB_GNTn is deasserted one clock cycle after SB_REQn is deasserted.

It is the responsibility of the external master to relinquish the Secondary Bus in time to prevent stalling of the L64364 core modules. Also, an SDRAM refresh will not pre-empt a Secondary Bus transaction, so the refresh timer must be set to accommodate maximum external bus master transactions.

Figure 10.26 Secondary Bus Grant Timing

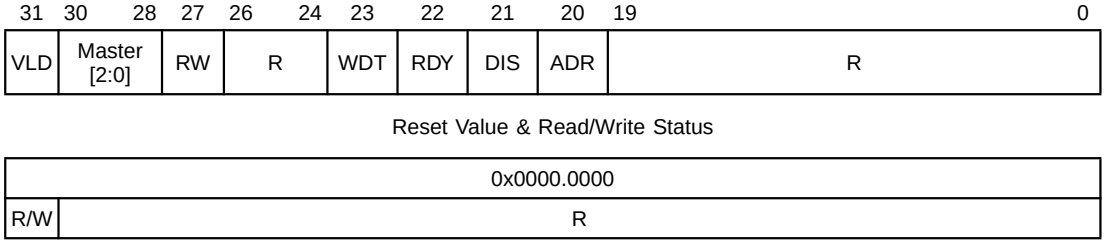


10.10 Error Reporting

Two registers aid in error debugging. The SB_Err (Secondary Bus Error) register at 0xB800.0820 contains information on the type of error and the offending master (see Figure 10.27). The SB_ErrAddr (Secondary Bus Error Address) register at 0xB800.0824 contains the address associated with the errored access (see Figure 10.28).

These registers capture information on the first error received and contain valid data when the VLD bit in the SB_Err register is set. The APU re-enables the capturing of future errors by clearing the VLD bit.

Figure 10.27 SB_Err Register



VLD **Secondary Bus Error Valid Bit** **31**

The VLD bit is set when the SB_Err register has captured data pertaining to a Secondary Bus error condition. VLD is cleared by writing a 1 to the bit position. Clearing the VLD bit enables the capturing of data pertaining to the next Secondary Bus error condition.

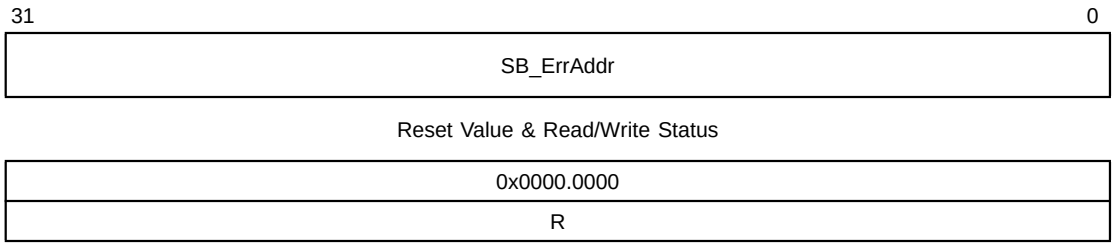
Master[2:0] **Secondary Bus Master Field** **[30:28]**

The Master that caused the Secondary Bus Error Condition.

Master[2:0]	Clock Delay
0x0	APU
0x1	TMU
0x2	PCI
0x3	MOVE
0x4	EDMA
0x5	EDMA
0x6	SB_REQn
0x7	Reserved

R/W	**Read/Write Indicator** When R/W is set, the Secondary Bus error occurred on a read access. When R/W is cleared, the Secondary Bus error occurred on a write access.	**27**
R	**Reserved** Not used in the L64364.	**[26:24]**
WDT	**WatchDog Time-Out** When set, indicates that the access caused an APU WatchDog time-out.	**23**
RDY	**SB_RDY Time-Out** When set, indicates that the Secondary Bus error is due to SB_RDYn not being returned within 64 clocks from the start of the access.	**22**
DIS	**Disabled Memory Page Access** When set, indicates that the Secondary Bus error is due to an access to a disabled Secondary Bus memory page.	**21**
ADR	**Address Error** When set, indicates that the Secondary Bus error is due to an access to an undefined Secondary Bus address (addresses not defined in Table 10.1 and Table 10.2).	**20**
R	**Reserved** Not used in the L64364.	**[19:0]**

Figure 10.28 SB_ErrAddr Register



Chapter 11
System Clock

This chapter describes the L64364 System Clock module and includes the following sections:

- Section 11.1, “System Clock Options,” page 11-1
- Section 11.2, “Clock Synthesis,” page 11-2
- Section 11.3, “Design Considerations,” page 11-4

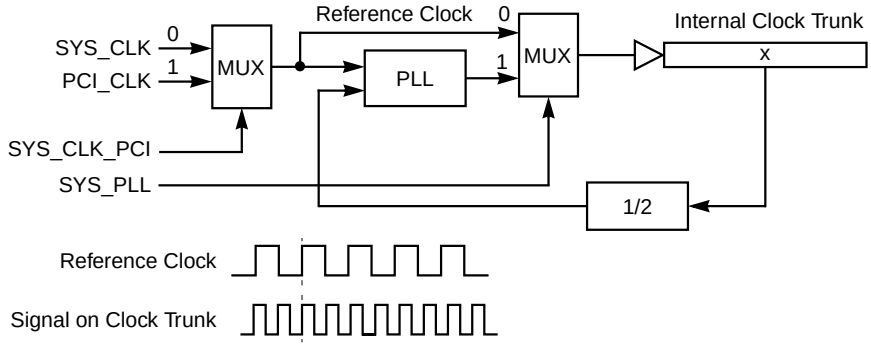
11.1 System Clock Options

The L64364 provides two options for selecting an input source for the System Clock. The two input options include:

- PCI Clock (PCI_CLK)
- System Clock (SYS_CLK)

Figure 11.1 shows a block diagram of the Clock Selection and Synthesis Circuit. When asserted, SYS_CLK_PCI selects PCI_CLK as the clock source for the L64364.

Figure 11.1 Clock Selection and Synthesis Circuit



The L64364 integrates a Phase-Locked Loop (PLL) to optionally synthesize an internal clock that is twice the frequency of PCI_CLK or SYS_CLK. The PLL operates at 15–100 MHz. The output of the PLL is selected by asserting the SYS_PLL signal. SYS_PLL must be stable from power up. Therefore, it is recommended that SYS_PLL be tied either to power (V_{DD}) or ground (V_{SS}).

On power up, it is recommended that the system clock source be stabilized at the operating frequency before deasserting PCI_RSTn.

When PCI_CLK is used as the system clock source, tie SYS_CLK either to power (V_{DD}) or ground (V_{SS}).

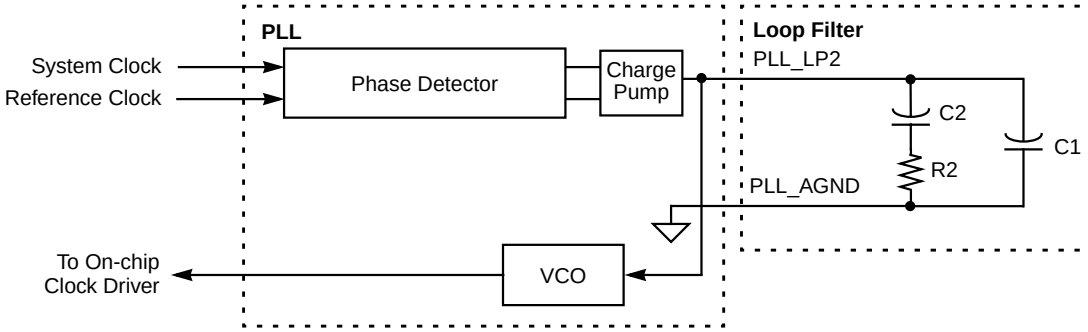
11.2 Clock Synthesis

The PLL (Figure 11.2) consists of a phase detector, charge pump, external loop filter, and voltage-controlled oscillator (VCO). The phase detector monitors the phase difference between the incoming reference clock and the output of the VCO divided by 2. At startup, the reference clock frequency is greater than one-half the VCO frequency causing the voltage at the output of the charge pump to increase. This, in turn, causes the VCO to increase in frequency.

When the VCO frequency reaches twice the reference clock frequency, the input signals to the phase detector are the same frequency and the output of the phase detector represents their phase difference. The phase detector output then speeds up and slows down the VCO to keep the internal System Clock's phase locked to that of the input reference clock.

Note: This circuit introduces significant skew between the clock inputs and the internal clock. Therefore, signals interfacing to the Secondary Bus or APU should be synchronized to the SB_CLKO output. Signals on the PCI Interface or Utopia Interface should be synchronized to the PCI_CLK or the Utopia TX and Rx clocks, respectively.

Figure 11.2 Phase-Locked Loop



You must provide the external loop filter. Table 11.1 lists the component values for the filter. The filter must be connected within one inch of the L64364's PLL_LP2 and PLL_AGND pins.

Table 11.1 Loop Filter Components

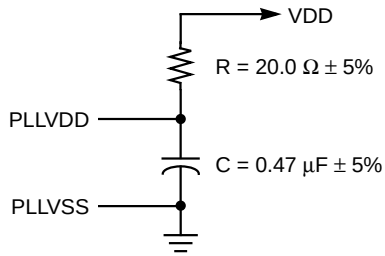
Component	Value	Tolerance
R2	200 Ω	±5%
C2	10 nF	±5%
C1	15–25 pF	±10%

11.3 Design Considerations

The following should be considered when designing the L64364 into a system:

- Phase tolerance and jitter are independent of PLL frequency. Jitter is affected by the noise frequency on the analog V_{DD} and V_{SS} lines. Jitter increases with the noise level on those lines so the noise level on the PLLVDD line should be kept under 10 mV. The level can be minimized with the following techniques:
  - Use wide PCB traces for the analog V_{DD} and V_{SS} connections to the PLL interface.
  - Use proper V_{DD} and V_{SS} decoupling.
  - Use good power and ground sources on the PC board.
  - In some setups, an RC network (see Figure 11.3) between the analog PLLVDD and PLLVSS supplies reduces jitter by filtering the noise on the lines.
- Never connect the PLL_AGND pin to the PC board ground. The PLL_AGND pin should be connected only to the external PLL filter as shown in Figure 11.2.
- An LVCMOS- or LVTTL-level input reference clock is recommended for signal compatibility with the PLL block. Other levels, such as 5 V CMOS or TTL, may degrade the tolerances.

Figure 11.3 PLL Supply Filtering



Chapter 12
JTAG Interface

This chapter describes the L64364 JTAG Interface. It assumes the reader is familiar with the *IEEE 1149.1, Standard Test Access Port and Boundary Scan Architecture* (JTAG) specification, and only describes the JTAG instructions supported by the L64364 and the bit order of the L64364 boundary scan chain. This chapter includes the following sections:

- Section 12.1, “JTAG Instructions,” page 12-1
- Section 12.2, “Boundary Scan Chain Order,” page 12-3

12.1 JTAG Instructions

The L64364 contains a 3-bit instruction register supporting the BYPASS, SAMPLE/PRLOAD, EXTEST, and HI_Z instructions. The MSB of the register is connected to the JTAG_TDI Input signal. Table 12.1 defines the bit encoding of the L64364 JTAG Instruction register.

Table 12.1 JTAG Instruction Register Encoding

Inst_reg[2:0]	Instruction
0b000	EXTEST
0b001	SAMPLE/PRELOAD
0b010	Reserved (EXTEST)
0b011	Reserved (BYPASS)
0b100	Reserved (BYPASS)
0b101	Reserved (BYPASS)
0b110	HI-Z
0b111	BYPASS

12.1.1 BYPASS Instruction

The BYPASS instruction selects the one-bit bypass register between the JTAG_TDI and JTAG_TDO pins. L64364 I/O pins are left in their normal functional mode.

12.1.2 SAMPLE/PRELOAD Instruction

The SAMPLE/PRELOAD instruction selects the L64364 boundary scan chain between the JTAG_TDI and JTAG_TDO pins. L64364 I/O pins are in their normal functional mode. The SAMPLE/PRELOAD instruction can be used to sample and scan out the current state of the I/O pins, or it can be used to scan data into the boundary scan chain in preparation for the EXTEST instruction. The SAMPLE/PRELOAD instruction has no affect on the normal operation of the L64364.

12.1.3 EXTEST Instruction

The EXTEST instruction selects the L64364 boundary scan chain between the JTAG_TDI and JTAG_TDO pins. EXTEST drives the data loaded in the boundary scan chain onto L64364 outputs based on the settings of the 3-state control bits in the boundary scan chain. If a 3-state control bit is cleared, its corresponding outputs are enabled to drive the pin. If a 3-state control bit is set, its corresponding outputs will be 3-stated.

12.1.4 HI-Z Instruction

The HI-Z instruction 3-states all L64364 output and bidirectional signals. The bypass register between the JTAG_TDI and JTAG_TDO pins is selected.

12.2 Boundary Scan Chain Order

Table 12.2 shows the bit order of the L64364 boundary scan chain.

Table 12.2 L64364 Boundary Scan Chain

Boundary Scan Chain Order	Signal	Comment
JTAG_TDI	JTAG_TDI	JTAG_TDI input pin
bscn_chain[335]	TM_CLK	Input
bscn_chain[334]	TEST_EN	Input
bscn_chain[333]	SCAN_EN	Input
bscn_chain[332]	TX_ADDR[0]	Input
bscn_chain[331]	TX_ADDR[0]	Output
bscn_chain[330]	TX_ADDR[1]	Input
bscn_chain[329]	TX_ADDR[1]	Output
bscn_chain[328]	TX_ADDR[2]	Input
bscn_chain[327]	TX_ADDR[2]	Output
bscn_chain[326]	TX_ADDR[3]	Input
bscn_chain[325]	TX_ADDR[3]	Output
bscn_chain[324]	TX_ADDR[4]	Input
bscn_chain[323]	TX_ADDR[4]	Output
bscn_chain[322]	TX_DATA[0]	Input
bscn_chain[321]	TX_DATA[0]	Output
(Sheet 1 of 15)		

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[320]	TX_DATA[1]	Input
bscn_chain[319]	TX_DATA[1]	Output
bscn_chain[318]	TX_DATA[2]	Input
bscn_chain[317]	TX_DATA[2]	Output
bscn_chain[316]	TX_DATA[3]	Input
bscn_chain[315]	TX_DATA[3]	Output
bscn_chain[314]	TX_DATA[4]	Input
bscn_chain[313]	TX_DATA[4]	Output
bscn_chain[312]	TX_DATA[5]	Input
bscn_chain[311]	TX_DATA[5]	Output
bscn_chain[310]	TX_DATA[6]	Input
bscn_chain[309]	TX_DATA[6]	Output
bscn_chain[308]	TX_DATA[7]	Input
bscn_chain[307]	TX_DATA[7]	Output
bscn_chain[306]	TX_PRTY	Input
bscn_chain[305]	3-State control bit[1]	3-State control for TX_DATA[7:0], TX_SOC, TX_PRTY
bscn_chain[304]	TX_PRTY	Output
bscn_chain[303]	TX_ENBn	Input
bscn_chain[302]	TX_ENBn	Output
bscn_chain[301]	TX_SOC	Input
bscn_chain[300]	TX_SOC	Output
bscn_chain[299]	TX_CLAV_0	Input
bscn_chain[298]	3-State control bit[1]	3-State control for TX_CLAV_0

(Sheet 2 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[297]	TX_CLAV_0	Output
bscn_chain[296]	TX_CLAV[1]	Input
bscn_chain[295]	TX_CLAV[2]	Input
bscn_chain[294]	TX_CLAV[3]	Input
bscn_chain[293]	TX_CLK	Input
bscn_chain[292]	RX_CLK	Input
bscn_chain[291]	RX_CLAV[3]	Input
bscn_chain[290]	RX_CLAV[2]	Input
bscn_chain[289]	RX_CLAV[1]	Input
bscn_chain[288]	RX_CLAV_0	Input
bscn_chain[287]	3-State control bit[1]	3-State control for RX_CLAV_0
bscn_chain[286]	RX_CLAV_0	Output
bscn_chain[285]	RX_ADDR[0]	Input
bscn_chain[284]	3-State control bit[1]	3-State control for TX_ENBn, TX_ADDR[4:0], RX_ENBn, RX_ADDR[4:0]
bscn_chain[283]	RX_ADDR[0]	Output
bscn_chain[282]	RX_ADDR[1]	Input
bscn_chain[281]	RX_ADDR[1]	Output
bscn_chain[280]	RX_ADDR[2]	Input
bscn_chain[279]	RX_ADDR[2]	Output
bscn_chain[278]	RX_ADDR[3]	Input
bscn_chain[277]	RX_ADDR[3]	Output
bscn_chain[276]	RX_ADDR[4]	Input

(Sheet 3 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[275]	RX_ADDR[4]	Output
bscn_chain[274]	RX_DATA[0]	Input
bscn_chain[273]	RX_DATA[0]	Output
bscn_chain[272]	RX_DATA[1]	Input
bscn_chain[271]	RX_DATA[1]	Output
bscn_chain[270]	RX_DATA[2]	Input
bscn_chain[269]	RX_DATA[2]	Output
bscn_chain[268]	RX_DATA[3]	Input
bscn_chain[267]	RX_DATA[3]	Output
bscn_chain[266]	RX_DATA[4]	Input
bscn_chain[265]	RX_DATA[4]	Output
bscn_chain[264]	RX_DATA[5]	Input
bscn_chain[263]	RX_DATA[5]	Output
bscn_chain[262]	RX_DATA[6]	Input
bscn_chain[261]	RX_DATA[6]	Output
bscn_chain[260]	RX_DATA[7]	Input
bscn_chain[259]	RX_DATA[7]	Output
bscn_chain[258]	RX_PRTY	Input
bscn_chain[257]	3-State control bit[1]	3-State control for RX_DATA[7:0], RX_SOC, RX_PRTY
bscn_chain[256]	RX_PRTY	Output
bscn_chain[255]	RX_ENBn	Input
bscn_chain[254]	RX_ENBn	Output
bscn_chain[253]	RX_SOC	Input

(Sheet 4 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[252]	RX_SOC	Output
bscn_chain[251]	PCI_INTn	PCI_INTn is only controlled by its 3-state control pin, and will be either 3-state or driving ‘0’
bscn_chain[250]	PCI_RSTn	Input
bscn_chain[249]	PCI_GNTn	Input
bscn_chain[248]	3-State control bit[1]	3-State control for PCI_REQn
bscn_chain[247]	PCI_REQn	Output
bscn_chain[246]	PCI_AD[31]	Input
bscn_chain[245]	3-State control bit[1]	3-State control for PCI_AD[31]
bscn_chain[244]	PCI_AD[31]	Output
bscn_chain[243]	PCI_AD[30]	Input
bscn_chain[242]	3-State control bit	3-State control for PCI_AD[30]
bscn_chain[241]	PCI_AD[30]	Output
bscn_chain[240]	PCI_AD[29]	Input
bscn_chain[239]	3-State control bit[1]	3-State control for PCI_AD[29]
bscn_chain[238]	PCI_AD[29]	Output
bscn_chain[237]	PCI_AD[28]	Input
bscn_chain[236]	3-State control bit[1]	3-State control for PCI_AD[28]
bscn_chain[235]	PCI_AD[28]	Output
bscn_chain[234]	PCI_AD[27]	Input
bscn_chain[233]	3-State control bit[1]	3-State control for PCI_AD[27]
bscn_chain[232]	PCI_AD[27]	Output
bscn_chain[231]	PCI_AD[26]	Input

(Sheet 5 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[230]	3-State control bit[1]	3-State control for PCI_AD[26]
bscn_chain[229]	PCI_AD[26]	Output
bscn_chain[228]	PCI_AD[25]	Input
bscn_chain[227]	3-State control bit[1]	3-State control for PCI_AD[25]
bscn_chain[226]	PCI_AD[25]	Output
bscn_chain[225]	PCI_AD[24]	Input
bscn_chain[224]	3-State control bit[1]	3-State control for PCI_AD[24]
bscn_chain[223]	PCI_AD[24]	Output
bscn_chain[222]	PCI_CBEn[3]	Input
bscn_chain[221]	3-State control bit[1]	3-State control for PCI_CBEn[3]
bscn_chain[220]	PCI_CBEn[3]	Output
bscn_chain[219]	PCI_IDSEL	Input
bscn_chain[218]	PCI_AD[23]	Input
bscn_chain[217]	3-State control bit[1]	3-State control for PCI_AD[23]
bscn_chain[216]	PCI_AD[23]	Output
bscn_chain[215]	PCI_AD[22]	Input
bscn_chain[214]	3-State control bit[1]	3-State control for PCI_AD[22]
bscn_chain[213]	PCI_AD[22]	Output
bscn_chain[212]	PCI_AD[21]	Input
bscn_chain[211]	3-State control bit[1]	3-State control for PCI_AD[21]
bscn_chain[210]	PCI_AD[21]	Output
bscn_chain[209]	PCI_AD[20]	Input
bscn_chain[208]	3-State control bit[1]	3-State control for PCI_AD[20]
bscn_chain[207]	PCI_AD[20]	Output

(Sheet 6 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[206]	PCI_AD[19]	Input
bscn_chain[205]	3-State control bit[1]	3-State control for PCI_AD[19]
bscn_chain[204]	PCI_AD[19]	Output
bscn_chain[203]	PCI_AD[18]	Input
bscn_chain[202]	3-State control bit[1]	3-State control for PCI_AD[18]
bscn_chain[201]	PCI_AD[18]	Output
bscn_chain[200]	PCI_AD[17]	Input
bscn_chain[199]	3-State control bit[1]	3-State control for PCI_AD[17]
bscn_chain[198]	PCI_AD[17]	Output
bscn_chain[197]	PCI_AD[16]	Input
bscn_chain[196]	3-State control bit[1]	3-State control for PCI_AD[16]
bscn_chain[195]	PCI_AD[16]	Output
bscn_chain[194]	PCI_CBEn[2]	Input
bscn_chain[193]	3-State control bit[1]	3-State control for PCI_CBEn[2]
bscn_chain[192]	PCI_CBEn[2]	Output
bscn_chain[191]	PCI_FRAME	Input
bscn_chain[190]	3-State control bit[1]	3-State control for PCI_FRAMEn
bscn_chain[189]	PCI_FRAMEn	Output
bscn_chain[188]	PCI_IRDYn	Input
bscn_chain[187]	3-State control bit[1]	3-State control for PCI_IRDYn
bscn_chain[186]	PCI_IRDYn	Output
bscn_chain[185]	PCI_TRDYn	Input
bscn_chain[184]	PCI_TRDYn	Output
bscn_chain[183]	PCI_DEVSELn	Input

(Sheet 7 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[182]	3-State control bit[1]	3-State control for PCI_DEVSELn, PCI_TRDYn, PCI_STOPn
bscn_chain[181]	PCI_DEVSELn	Output
bscn_chain[180]	PCI_STOPn	Input
bscn_chain[179]	PCI_STOPn	Output
bscn_chain[178]	PCI_PERRn	Input
bscn_chain[177]	3-State control bit[1]	3-State control for PCI_PERRn
bscn_chain[176]	PCI_PERRn	Output
bscn_chain[175]	PCI_SERRn	PCI_SERRn is only controlled by its 3-state control pin, and will be either 3-state or driving ‘0’
bscn_chain[174]	PCI_PAR	Input
bscn_chain[173]	3-State control bit[1]	3-State control for PCI_PAR
bscn_chain[172]	PCI_PAR	Output
bscn_chain[171]	PCI_CBEn[1]	Input
bscn_chain[170]	3-State control bit[1]	3-State control for PCI_CBEn[1]
bscn_chain[169]	PCI_CBEn[1]	Output
bscn_chain[168]	PCI_AD[15]	Input
bscn_chain[167]	3-State control bit[1]	3-State control for PCI_AD[15]
bscn_chain[166]	PCI_AD[15]	Output
bscn_chain[165]	PCI_AD[14]	Input
bscn_chain[164]	3-State control bit[1]	3-State control for PCI_AD[14]
bscn_chain[163]	PCI_AD[14]	Output
bscn_chain[162]	PCI_AD[13]	Input
bscn_chain[161]	3-State control bit[1]	3-State control for PCI_AD[13]

(Sheet 8 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[160]	PCI_AD[13]	Output
bscn_chain[159]	PCI_AD[12]	Input
bscn_chain[158]	3-State control bit[1]	3-State control for PCI_AD[12]
bscn_chain[157]	PCI_AD[12]	Output
bscn_chain[156]	PCI_AD[11]	Input
bscn_chain[155]	3-State control bit[1]	3-State control for PCI_AD[11]
bscn_chain[154]	PCI_AD[11]	Output
bscn_chain[153]	PCI_AD[10]	Input
bscn_chain[152]	3-State control bit[1]	3-State control for PCI_AD[10]
bscn_chain[151]	PCI_AD[10]	Output
bscn_chain[150]	PCI_AD[9]	Input
bscn_chain[149]	3-State control bit[1]	3-State control for PCI_AD[9]
bscn_chain[148]	PCI_AD[9]	Output
bscn_chain[147]	PCI_AD[8]	Input
bscn_chain[146]	3-State control bit[1]	3-State control for PCI_AD[8]
bscn_chain[145]	PCI_AD[8]	Output
bscn_chain[144]	PCI_CBEn[0]	Input
bscn_chain[143]	3-State control bit[1]	3-State control for PCI_CBEn[0]
bscn_chain[142]	PCI_CBEn[0]	Output
bscn_chain[141]	PCI_AD[7]	Input
bscn_chain[140]	3-State control bit[1]	3-State control for PCI_AD[7]
bscn_chain[139]	PCI_AD[7]	Output
bscn_chain[138]	PCI_AD[6]	Input
bscn_chain[137]	3-State control bit[1]	3-State control for PCI_AD[6]

(Sheet 9 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[136]	PCI_AD[6]	Output
bscn_chain[135]	PCI_AD[5]	Input
bscn_chain[134]	3-State control bit[1]	3-State control for PCI_AD[5]
bscn_chain[133]	PCI_AD[5]	Output
bscn_chain[132]	PCI_AD[4]	Input
bscn_chain[131]	3-State control bit[1]	3-State control for PCI_AD[4]
bscn_chain[130]	PCI_AD[4]	Output
bscn_chain[129]	PCI_AD[3]	Input
bscn_chain[128]	3-State control bit[1]	3-State control for PCI_AD[3]
bscn_chain[127]	PCI_AD[3]	Output
bscn_chain[126]	PCI_AD[2]	Input
bscn_chain[125]	3-State control bit[1]	3-State control for PCI_AD[2]
bscn_chain[124]	PCI_AD[2]	Output
bscn_chain[123]	PCI_AD[1]	Input
bscn_chain[122]	3-State control bit[1]	3-State control for PCI_AD[1]
bscn_chain[121]	PCI_AD[1]	Output
bscn_chain[120]	PCI_AD[0]	Input
bscn_chain[119]	3-State control bit[1]	3-State control for PCI_AD[0]
bscn_chain[118]	PCI_AD[0]	Output
bscn_chain[117]	SB_A[2]	Output
bscn_chain[116]	SB_A[3]	Output
bscn_chain[115]	SB_A[4]	Output
bscn_chain[114]	PCI_CLK	Input
bscn_chain[113]	SB_A[5]	Output

(Sheet 10 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[112]	SB_A[6]	Output
bscn_chain[111]	SB_A[7]	Output
bscn_chain[110]	SB_A[8]	Output
bscn_chain[109]	SB_A[9]	Output
bscn_chain[108]	SB_A[10]	Output
bscn_chain[107]	SB_A[11]	Output
bscn_chain[106]	SB_A[12]	Output
bscn_chain[105]	SB_A[13]	Output
bscn_chain[104]	SB_A[14]	Output
bscn_chain[103]	SB_A[15]	Output
bscn_chain[102]	SB_A[16]	Output
bscn_chain[101]	SB_A[17]	Output
bscn_chain[100]	SB_A[18]	Output
bscn_chain[99]	SB_A[19]	Output
bscn_chain[98]	SB_A[20]	Output
bscn_chain[97]	SB_A[21]	Output
bscn_chain[96]	3-State control bit[1]	3-State control for SB_A[21:2]
bscn_chain[95]	SB_D[0]	Input
bscn_chain[94]	SB_D[0]	Output
bscn_chain[93]	SB_D[1]	Input
bscn_chain[92]	SB_D[1]	Output
bscn_chain[91]	SB_D[2]	Input
bscn_chain[90]	SB_D[2]	Output
bscn_chain[89]	SB_D[3]	Input

(Sheet 11 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[88]	SB_D[3]	Output
bscn_chain[87]	SB_D[4]	Input
bscn_chain[86]	SB_D[4]	Output
bscn_chain[85]	SB_D[5]	Input
bscn_chain[84]	SB_D[5]	Output
bscn_chain[83]	SB_D[6]	Input
bscn_chain[82]	SB_D[6]	Output
bscn_chain[81]	SB_D[7]	Input
bscn_chain[80]	SB_D[7]	Output
bscn_chain[79]	SB_D[8]	Input
bscn_chain[78]	SB_D[8]	Output
bscn_chain[77]	SB_D[9]	Input
bscn_chain[76]	SB_D[9]	Output
bscn_chain[75]	SB_D[10]	Input
bscn_chain[74]	SB_D[10]	Output
bscn_chain[73]	SB_D[11]	Input
bscn_chain[72]	SB_D[11]	Output
bscn_chain[71]	SB_D[12]	Input
bscn_chain[70]	SB_D[12]	Output
bscn_chain[69]	SB_D[13]	Input
bscn_chain[68]	SB_D[13]	Output
bscn_chain[67]	SB_D[14]	Input
bscn_chain[66]	SB_D[14]	Output
bscn_chain[65]	SB_D[15]	Input

(Sheet 12 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[64]	SB_D[15]	Output
bscn_chain[63]	SB_D[16]	Input
bscn_chain[62]	SB_D[16]	Output
bscn_chain[61]	3-State control bit[1]	3-State control for SB_D[31:0]
bscn_chain[60]	SB_D[17]	Input
bscn_chain[59]	SB_D[17]	Output
bscn_chain[58]	SB_D[18]	Input
bscn_chain[57]	SB_D[18]	Output
bscn_chain[56]	SB_D[19]	Input
bscn_chain[55]	SB_D[19]	Output
bscn_chain[54]	SB_D[20]	Input
bscn_chain[53]	SB_D[20]	Output
bscn_chain[52]	SB_D[21]	Input
bscn_chain[51]	SB_D[21]	Output
bscn_chain[50]	SB_D[22]	Input
bscn_chain[49]	SB_D[22]	Output
bscn_chain[48]	SB_D[23]	Input
bscn_chain[47]	SB_D[23]	Output
bscn_chain[46]	SB_D[24]	Input
bscn_chain[45]	SB_D[24]	Output
bscn_chain[44]	SB_D[25]	Input
bscn_chain[43]	SB_D[25]	Output
bscn_chain[42]	SB_D[26]	Input
bscn_chain[41]	SB_D[26]	Output

(Sheet 13 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[40]	SB_D[27]	Input
bscn_chain[39]	SB_D[27]	Output
bscn_chain[38]	SB_D[28]	Input
bscn_chain[37]	SB_D[28]	Output
bscn_chain[36]	SB_D[29]	Input
bscn_chain[35]	SB_D[29]	Output
bscn_chain[34]	SB_D[30]	Input
bscn_chain[33]	SB_D[30]	Output
bscn_chain[32]	SB_D[31]	Input
bscn_chain[31]	SB_D[31]	Output
bscn_chain[30]	3-State control bit[1]	3-State control for SB_PCSn[4:0], SB_OEn[3:0], SB_WEn[3:0]
bscn_chain[29]	SB_WEn[0]	Output
bscn_chain[28]	SB_WEn[1]	Output
bscn_chain[27]	SB_WEn[2]	Output
bscn_chain[26]	SB_WEn[3]	Output
bscn_chain[25]	SB_OEn[0]	Output
bscn_chain[24]	SB_OEn[1]	Output
bscn_chain[23]	SB_OEn[2]	Output
bscn_chain[22]	SB_OEn[3]	Output
bscn_chain[21]	SB_PCSn[0]	Output
bscn_chain[20]	SB_PCSn[1]	Output
bscn_chain[19]	SB_PCSn[2]	Output
bscn_chain[18]	SB_PCSn[3]	Output

(Sheet 14 of 15)

Table 12.2 L64364 Boundary Scan Chain (Cont.)

Boundary Scan Chain Order	Signal	Comment
bscn_chain[17]	SB_PCSn[4]	Output
bscn_chain[16]	SB_GNTn	Output
bscn_chain[15]	SB_RDYn	Input
bscn_chain[14]	SB_REQn	Input
bscn_chain[13]	SE_ACK	Input
bscn_chain[12]	SE_CLK	Output
bscn_chain[11]	SE_DI	Input
bscn_chain[10]	SYS_CLK_PCI	Input
bscn_chain[9]	SYS_CLK	Input
bscn_chain[8]	SYS_PLL	Input
bscn_chain[7]	SYS_BOOT[0]	Input
bscn_chain[6]	SYS_BOOT[1]	Input
bscn_chain[5]	SYS_CPCOND	Input
bscn_chain[4]	SYS_INTn[0]	Input
bscn_chain[3]	SYS_INTn[1]	Input
bscn_chain[2]	SYS_NMIn	Input
bscn_chain[1]	SYS_OE	Input
bscn_chain[0]	SYS_PSTALLn	Output
JTAG_TDO	JTAG_TDO	JTAG_TDO output pin
(Sheet 15 of 15)		

1. These bits control 3-state outputs during the EXTEST.

Chapter 13
Specifications

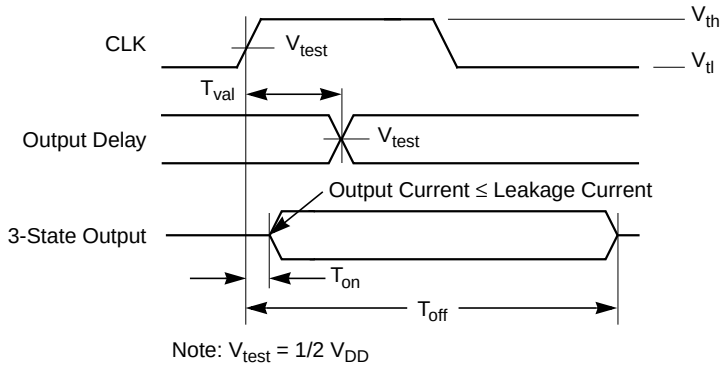
This chapter describes the electrical specifications for the LSI Logic L64364 chip. The following sections are included:

- Section 13.1, “AC Timing,” page 13-1
- Section 13.2, “Electrical Requirements,” page 13-10
- Section 13.3, “Pin Summary,” page 13-14
- Section 13.4, “Package Information,” page 13-17

13.1 AC Timing

This section specifies the AC timing characteristics of the L64364. Figure 13.1 and Figure 13.2 identify reference points for output and input signal timings, respectively. Table 13.1 through Table 13.5 contain AC timing information.

Figure 13.1 Output Signal Timing Reference Points



Note: V_{test} = 1/2 V_{DD}

Figure 13.2 Input Signal Timing Reference Points

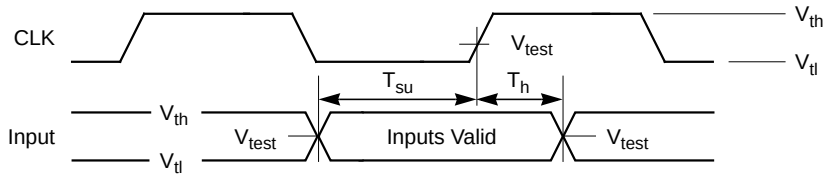


Table 13.1 PCI Interface Timing

Ref #	Signal Timing	Reference Clock	Min	Max	Units
P1	PCI_CLK cycle time	–	30	–	ns
P2	PCI_CLK HIGH time	–	11	–	ns
P3	PCI_CLK LOW time	–	11	–	ns
P4	PCI_CLK slew rate	–	1	4	V/ns
P5	PCI_RSTn slew rate	–	50	–	mV/ns
P6	PCI_AD[31:0] input setup time	PCI_CLK	7	–	ns
P7	PCI_AD[31:0] input hold time	PCI_CLK	0	–	ns
P8	PCI_AD[31:0] output valid	PCI_CLK	1	11	ns
	PCI_AD[31:0] output float	PCI_CLK	–	10	ns
p9	PCI_CBE[3:0] input setup time	PCI_CLK	7	–	ns
p10	PCI_CBE[3:0] input hold time	PCI_CLK	0	–	ns
p11	PCI_CBE[3:0] output valid	PCI_CLK	2	11	ns
	PCI_CBE[3:0] output float	PCI_CLK	–	10	ns
p12	PCI_PAR input setup time	PCI_CLK	7	–	ns
p13	PCI_PAR input hold time	PCI_CLK	0	–	ns
p14	PCI_PAR output valid	PCI_CLK	2	11	ns
	PCI_PAR output float	PCI_CLK	–	10	ns
p15	PCI_FRAMEn input setup time	PCI_CLK	7	–	ns
P16	PCI_FRAMEn input hold time	PCI_CLK	0	–	ns

(Sheet 1 of 3)

Table 13.1 PCI Interface Timing (Cont.)

Ref #	Signal Timing	Reference Clock	Min	Max	Units
P17	PCI_FRAMEn output valid	PCI_CLK	2	11	ns
	PCI_FRAMEn output float	PCI_CLK	–	10	ns
P18	PCI_IRDYn input setup time	PCI_CLK	7	–	ns
P19	PCI_IRDYn input hold time	PCI_CLK	0	–	ns
P20	PCI_IRDYn output valid	PCI_CLK	2	11	ns
	PCI_IRDYn output float	PCI_CLK	–	10	ns
P21	PCI_TRDYn input setup time	PCI_CLK	7	–	ns
P22	PCI_TRDYn input hold time	PCI_CLK	0	–	ns
P23	PCI_TRDYn output valid	PCI_CLK	2	11	ns
	PCI_TRDYn output float	PCI_CLK	–	10	ns
P24	PCI_STOPn input setup time	PCI_CLK	7	–	ns
P25	PCI_STOPn input hold time	PCI_CLK	0	–	ns
P26	PCI_STOPn output valid	PCI_CLK	2	11	ns
	PCI_STOPn output float	PCI_CLK	–	10	ns
P27	PCI_IDSEL input setup time	PCI_CLK	7	–	ns
P28	PCI_IDSEL input hold time	PCI_CLK	0	–	ns
P29	PCI_DEVSELn input setup time	PCI_CLK	7	–	ns
P30	PCI_DEVSELn input hold time	PCI_CLK	0	–	ns
P31	PCI_DEVSELn output valid	PCI_CLK	2	11	ns
	PCI_DEVSELn output float	PCI_CLK	–	10	ns
P32	PCI_REQn output valid	PCI_CLK	2	12	ns
P33	PCI_GNTn input setup time	PCI_CLK	10	–	ns
P34	PCI_GNTn input hold time	PCI_CLK	0	–	ns
P35	PCI_PERRn input setup time	PCI_CLK	7	–	ns

(Sheet 2 of 3)

Table 13.1 PCI Interface Timing (Cont.)

Ref #	Signal Timing	Reference Clock	Min	Max	Units
P36	PCI_PERRn input hold time	PCI_CLK	0	–	ns
P37	PCI_PERRn output valid	PCI_CLK	2	11	ns
	PCI_PERRn output float	PCI_CLK	–	10	ns
P38	PCI_SERRn output valid	PCI_CLK	2	11	ns
	PCI_SERRn output float	PCI_CLK	–	10	ns
P39	PCI_INTn output valid, asynchronous	PCI_CLK	N/A	N/A	ns

(Sheet 3 of 3)

Secondary Bus best-case timings (minimum output delay) were simulated at best-case operating conditions and a 15 pF Secondary Bus load. Worst-case timings (minimum setup, maximum output delay) were simulated at worst-case operating conditions and an 85 pF load.

These timings are based on the SB_CLKO output clock and internal SB_DCLK (SB_D input capture clock) being within ± 0.5 ns of the internal system clock. See Section 10.2.2, "Secondary Bus Clock Control Register," for an explanation of the clock relationships. The default settings of the Secondary Bus Clock Control register meet the above requirement for SB_CLKO loadings from 15 pF to 65 pF.

Table 13.2 Secondary Bus Timing

Ref #	Signal Timing	Reference Clock	Min		Max		Units
			PX80	PX100	PX80	PX100	
S0	SB_CLKO period	–	12.5	10.0	–	–	ns
S1	SB_CLKO duty cycle (SYS_PLL = 0)	–	40		60		%
	SB_CLKO duty cycle (SYS_PLL = 1)	–	45		55		%
S2	SB_CLKO phase delay (SYS_CLK_PCI = 1, SYS_PLL = 0)	PCI_CLK	4.1		14.7		ns
S3	SB_CLKO phase delay (SYS_CLK_PCI = 0, SYS_PLL = 0)	SYS_CLK	2.8		12.8		ns
S4	SB_D[31:0] input setup time	SB_CLKO	2.0	1.0	–	–	ns
S5	SB_D[31:0] input hold time	SB_CLKO	0.5		–	–	ns
S6	SB_D[31:0] output valid	SB_CLKO	3.0	2.5	8.5	5.0	ns
	SB_D[31:0] output hold time	SB_CLKO	1.0		–	–	ns
	SB_D[31:0] output float	SB_CLKO	–	–	5.5	5.0	ns
S7	SB_A[21:2] output valid	SB_CLKO	3.0	2.5	9.0	6.0	ns
	SB_A[21:2] output hold time	SB_CLKO	1.0		–	–	ns
	SB_A[21:2] output float	SB_CLKO	–	–	5.5	5.0	ns
S8	SB_WEn[3:0] output valid	SB_CLKO	3.0	2.5	9.0	5.5	ns
	SB_WEn[3:0] output hold time	SB_CLKO	1.0		–	–	ns
	SB_WEn[3:0] output float	SB_CLKO	–	–	5.5	5.0	ns
S9	SB_OEn[3:0] output valid	SB_CLKO	3.0	2.5	9.0	5.5	ns
	SB_OEn[3:0] output hold time	SB_CLKO	1.0		–	–	ns
	SB_OEn[3:0] output float	SB_CLKO	–	–	5.5	5.0	ns

(Sheet 1 of 2)

Table 13.2 Secondary Bus Timing (Cont.)

Ref #	Signal Timing	Reference Clock	Min		Max		Units
			PX80	PX100	PX80	PX100	
S10	SB_PCSn[4:0] output valid	SB_CLKO	3.0	2.5	9.0	5.5	ns
	SB_PCSn[4:0] output hold time	SB_CLKO	1.0		–	–	ns
	SB_PCSn[4:0] output float	SB_CLKO	–	–	5.5	5.0	ns
S11	SB_REQn input setup time	SB_CLKO	2.0		–	–	ns
S12	SB_REQn input hold time	SB_CLKO	0.5		–	–	ns
S13	SB_GNTn output valid	SB_CLKO	4.0	3.0	10.5	8.0	ns
S14	SB_RDYn setup time[1]	SB_CLKO	2.0		–	–	–
S15	SB_RDYn hold time[1]	SB_CLKO	0.5		–	–	–

(Sheet 2 of 2)

1. While SB_RDYn is an asynchronous input, these times are provided if you choose to make the inputs synchronous.

Table 13.3 Utopia Interface Transmit Timing

Ref #	Signal Timing	Reference Clock	Min	Max	Units
U1	TX_CLK, RX_CLK frequency	–	0	50	MHz
U2	TX_CLK, RX_CLK duty cycle	–	40	60	%
U3	TX_DATA[7:0] input setup time	TX_CLK	4	–	ns
U4	TX_DATA[7:0] input hold time	TX_CLK	1	–	ns
U5	TX_DATA[7:0] output valid	TX_CLK	2	13	ns
	TX_DATA[7:0] output float	TX_CLK	–	6.5	ns
U6	TX_SOC input setup time	TX_CLK	4	–	ns
U7	TX_SOC input hold time	TX_CLK	1	–	ns

(Sheet 1 of 2)

Table 13.3 Utopia Interface Transmit Timing (Cont.)

Ref #	Signal Timing	Reference Clock	Min	Max	Units
U8	TX_SOC output valid	TX_CLK	2	10	ns
	TX_SOC output float	TX_CLK	–	6.5	ns
U9	TX_ENBn input setup time	TX_CLK	4	–	ns
U10	TX_ENBn input hold time	TX_CLK	1	–	ns
U11	TX_ENBn output valid	TX_CLK	2	10	ns
U12	TX_PRTY input setup time	TX_CLK	4	–	ns
U13	TX_PRTY input hold time	TX_CLK	1	–	ns
U14	TX_PRTY output valid	TX_CLK	2	15	ns
	TX_PRTY output float	TX_CLK	–	6.5	ns
U15	TX_ADDR[4:0] input setup time	TX_CLK	4	–	ns
U16	TX_ADDR[4:0] input hold time	TX_CLK	1	–	ns
U17	TX_ADDR[4:0] output valid	TX_CLK	2	10	ns
U18	TX_CLAV0 input setup time	TX_CLK	4	–	ns
U19	TX_CLAV0 input hold time	TX_CLK	1	–	ns
U20	TX_CLAV0 output valid	TX_CLK	2	13	ns
	TX_CLAV0 output float	TX_CLK	–	6.5	ns
U21	TX_CLAV[3:1] input setup time	TX_CLK	4	–	ns
U22	TX_CLAV[3:1] input hold time	TX_CLK	1	–	ns

(Sheet 2 of 2)

Table 13.4 Utopia Interface Receive Timing

Ref #	Signal Timing	Reference Clock	Min	Max	Units
U23	RX_DATA[7:0] input setup time	RX_CLK	4	–	ns
U24	RX_DATA[7:0] input hold time	RX_CLK	1	–	ns
U25	RX_DATA[7:0] output valid	RX_CLK	2	13	ns
	RX_DATA[7:0] output float	RX_CLK	–	6.5	ns
U26	RX_SOC input setup time	RX_CLK	4	–	ns
U27	RX_SOC input hold time	RX_CLK	1	–	ns
U28	RX_SOC output valid	RX_CLK	2	10	ns
	RX_SOC output float	RX_CLK	–	6.5	ns
U29	RX_ENBn input setup time	RX_CLK	4	–	ns
U30	RX_ENBn input hold time	RX_CLK	1	–	ns
U31	RX_ENBn output valid	RX_CLK	2	13	ns
U32	RX_PRTY input setup time	RX_CLK	4	–	ns
U33	RX_PRTY input hold time	RX_CLK	1	–	ns
U34	RX_PRTY output valid	RX_CLK	2	15	ns
	RX_PRTY output float	RX_CLK	–	6.5	ns
U35	RX_ADDR[4:0] input setup time	RX_CLK	4	–	ns
U36	RX_ADDR[4:0] input hold time	RX_CLK	1	–	ns
U37	RX_ADDR[4:0] output valid	RX_CLK	2	13	ns
U38	RX_CLAV0 input setup time	RX_CLK	4	–	ns
U39	RX_CLAV0 input hold time	RX_CLK	1	–	ns
U40	RX_CLAV0 output valid	RX_CLK	2	10	ns
	RX_CLAV0 output float	RX_CLK	–	6.5	ns
U41	RX_CLAV[3:1] input setup time	RX_CLK	4	–	ns
U42	RX_CLAV[3:1] input hold time	RX_CLK	1	–	ns

Table 13.5 Miscellaneous Timing

Ref #	Signal Timing	Reference Clock	Min		Max		Units
			PX80	PX100	PX80	PX100	
M1	SYS_CLK frequency (SYS_PLL = 0)	–	–	–	80.0	100.0	MHz
M2	SYS_CLK duty cycle (SYS_PLL = 0)	–	40		60		%
M3	SYS_CLK frequency (SYS_PLL = 1)	–	7.5		40.0	50.0	MHz
M4	SYS_CLK duty cycle (SYS_PLL = 1)	–	40		60		%
M5	SE_DI input setup time	SE_CLK	10		–	–	ns
M6	SE_DI input hold time	SE_CLK	5		–	–	ns
M7	SE_ACK input setup time	SE_CLK	10		–	–	ns
M8	SE_ACK input hold time	SE_CLK	5		–	–	ns
M9	JTAG_TCLK frequency	–	0		20		MHz
M10	JTAG_TCLK duty cycle	–	40		60		%
M11	JTAG_TDI setup time	JTAG_TCLK	1		–	–	ns
M12	JTAG_TDI hold time	JTAG_TCLK	3		–	–	ns
M13	JTAG_TM setup time	JTAG_TCLK	1		–	–	ns
M14	JTAG_TM hold time	JTAG_TCLK	3		–	–	ns
M15	JTAG_TDO output delay	JTAG_TCLK	7		15		ns
M16	JTAG_TRSTn pulse width	JTAG_TCLK	20		–	–	ns

13.2 Electrical Requirements

This section specifies the electrical requirements for the L64364.

13.2.1 I/O Pad Drivers and Receivers

Table 13.6 identifies the circuits used to drive signals to and receive signals from the L64364's input/output signal pins. The L64364 is manufactured in the LSI Logic G10®-p process technology.

Table 13.6 I/O Pad Drivers and Receivers

Pin Name	Pin Type	Internal Resistor	Drive Strength	Cell Type
JTAG_TCLK	I	pull-up	–	ibufuf
JTAG_TDI	I	pull-up	–	ibufuf
JTAG_TDO	O	–	4 mA	proc_drv
JTAG_TM	I	pull-up	–	ibufuf
JTAG_TRSTn	I	pull-up	–	ibufuf
PCI_AD[21:0]	I/O	–	PCI	rbdepci25f
PCI_CBEn[3:0]	I/O	–	PCI	rbdepci25f
PCI_CLK	I	–	–	ibuff
PCI_DEVSELn	I/O	–	PCI	rbdepci25f
PCI_FRAMEn	I/O	–	PCI	rbdepci25f
PCI_GNTn	I	–	PCI	rbdepci25f
PCI_IDSEL	I	–	PCI	rbdepci25f
PCI_INTn	I/O	–	PCI	rbdepci25f
PCI_IRDYn	I/O	–	PCI	rbdepci25f
PCI_PAR	I/O	–	PCI	rbdepci25f
PCI_PERRn	I/O	–	PCI	rbdepci25f
(Sheet 1 of 3)				

Table 13.6 I/O Pad Drivers and Receivers (Cont.)

Pin Name	Pin Type	Internal Resistor	Drive Strength	Cell Type
PCI_REQn	I	–	PCI	rbdepci25f
PCI_RSTn	I	–	PCI	rbdepci25f
PCI_SERRn	O	–	PCI	rbdepci25f
PCI_STOPn	I/O	–	PCI	rbdepci25f
PCI_TRDYn	I/O	–	PCI	rbdepci25f
PLL_AGND	I	–	–	PLL
PLL_LP2	I/O	–	–	PLL
PLL_IDDTn	I	pull-down	–	iiddtnf
RX_ADDR[4:0]	I/O	–	6 mA	bd6cf
RX_CLAV[3:0]	I/O	–	6 mA	bd6cf
RX_CLK	I	–	–	ibuff
RX_DATA[7:0]	I/O	–	6 mA	bd6cf
RX_ENBn	I/O	–	6 mA	bd6cf
RX_PRTY	I/O	–	6 mA	bd6cf
RX_SOC	I/O	–	6 mA	bd6cf
SB_A[21:2]	O	–	8 mA	bt8rp
SB_CLKO	O	–	12 mA	bt12
SB_D[31:0]	I/O	–	8 mA	bd8cf
SB_GNTn	O	–	8 mA	bt8rp
SB_OEn[3:0]	O	–	8 mA	bt8rp
SB_PCSn[4:0]	O	–	8 mA	bt8rp
SB_RDYn	I	pull-up	–	ibufuf
SB_REQn	I	pull-up	–	ibufuf
SB_WEn[3:0]	O	–	8 mA	bt8rp

(Sheet 2 of 3)

Table 13.6 I/O Pad Drivers and Receivers (Cont.)

Pin Name	Pin Type	Internal Resistor	Drive Strength	Cell Type
SCAN_EN	I	pull-down	–	ibufdf
SE_ACK	I	–	–	ibuff
SE_CLK	O	–	8 mA	bt8rp
SE_DI	I	–	–	ibuff
SYS_BOOT[1:0]	I	–	–	ibuff
SYS_CLK	I	–	–	ibuff
SYS_CLK_PCI	I	–	–	ibuff
SYS_CPCOND	I	–	–	ibuff
SYS_INTn[1:0]	I	–	–	ibuff
SYS_NMIn	I	–	6 mA	bd6cf
SYS_OE	I	–	–	icptnuf
SYS_PLL	I	–	–	ibuff
SYS_PSTALLn	O	–	8 mA	bt8rp
TEST_EN	I	pull-down	–	ibufdf
TM_CLK	I	–	–	ibuff
TX_ADDR[4:0]	I/O	–	6 mA	bd6cf
TX_CLAV[3:1]	I	–	–	ibuff
TX_CLAV[0]	I/O	–	6 mA	bd6cf
TX_CLK	I	–	–	ibuff
TX_DATA[7:0]	I/O	–	6 mA	bd6cf
TX_ENBn	I/O	–	6 mA	bd6cf
TX_PRTY	I/O	–	6 mA	bd6cf
TX_SOC	I/O	–	6 mA	bd6cf

(Sheet 3 of 3)

13.2.2 I/O Level Requirements

This section identifies the power and signal level characteristics of the L64364.

Table 13.7 DC Characteristics

Symbol	Parameter	Condition	Min	Typ	Max	Units
V_{DD}	Supply Voltage		3.135	3.3	3.465	V
V_{CC}	PCI clamp diode voltage		4.75	5.0	5.25	V
V_{ILP}	Voltage Input Low - PCI Bus		–0.5	–	0.8	V
V_{IHP}	Voltage Input High - PCI Bus		2.0	–	5.5	V
V_{OLP}	Voltage Output Low - PCI Bus		–	0.2	0.55	V
V_{OHP}	Voltage Output High - PCI Bus		2.4	–	V_{DD}	V
V_{ILS}	Voltage Input Low - Secondary Bus		–0.5	–	0.8	V
V_{IHS}	Voltage Input High - Secondary Bus		2.0	–	V_{DD} + 0.3	V
V_{OL}	Voltage Output Low - Secondary Bus		–	0.2	0.4	V
V_{OH}	Voltage Output High - Secondary Bus		2.4	–	V_{DD}	V
V_{IL}	Voltage Input Low - all other signals		–0.5	–	0.8	V
V_{IH}	Voltage Input High - all other signals		2.0	–	5.5	V
V_{OL}	Voltage Output Low - all other signals		–	0.2	0.4	V
V_{OH}	Voltage Output High - all other signals		2.4	–	V_{DD}	V
I_{IL}	Input current low	$V_{IN} = V_{SS}$	–10	–1	–	µA
I_{ILP}	Input current low - pins with pull-ups	$V_{IN} = V_{SS}$	–214	–115	–35	µA
I_{IH}	Input current high	$V_{IN} = V_{DD}$	–	1	10	µA
I_{IHP}	Input current high - pins with pull-downs	$V_{IN} = V_{DD}$	35	115	222	µA

Table 13.7 DC Characteristics (Cont.)

Symbol	Parameter	Condition	Min	Typ	Max	Units
I_{OZ}	3-state leakage current	$V_{IN} = V_{SS} - V_{DD}$	–10	–	10	µA
I_{DD}	Dynamic supply current	80 MHz operation	–	740	830	mA
		100 MHz operation	–	925	1040	mA

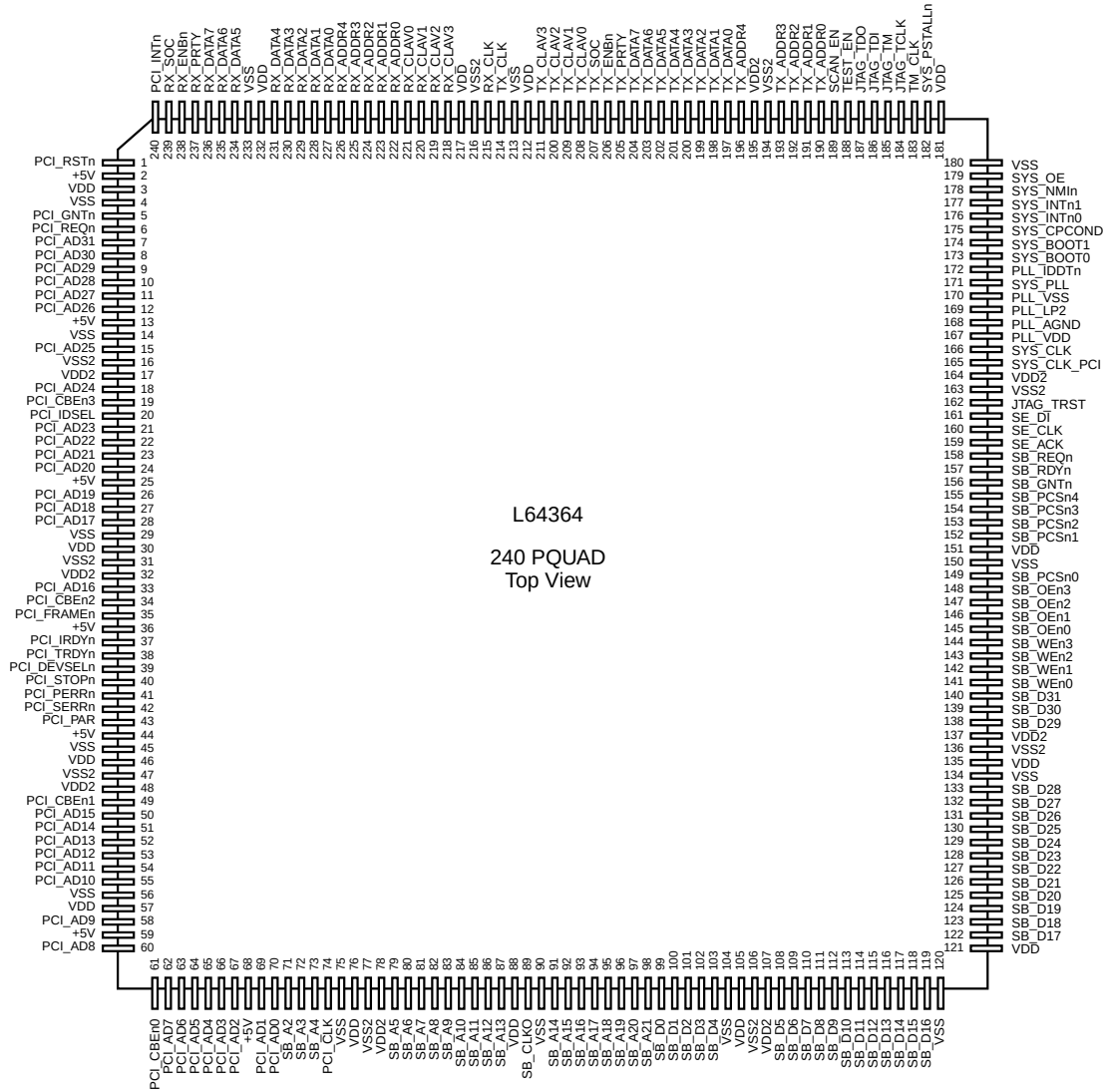
13.3 Pin Summary

Table 13.8 summarizes the L64364's input/output signal-to-pin assignments. The table is in alphabetical signal name order. Figure 13.4 shows the L64364 pinout.

Table 13.8 L64364 Pin Summary 240 Pin Alphabetical Pin List

Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal	Pin
JTAG_TCLK	184	PCI_PERRn	41	SB_A19	96	SB_WEn1	142	VDD	46
JTAG_TDI	186	PCI_REQn	6	SB_A20	97	SB_WEn2	143	VDD	57
JTAG_TDO	187	PCI_RSTn	1	SB_A21	98	SB_WEn3	144	VDD	76
JTAG_TM	185	PCI_SERRn	42	SB_CLKO	89	SCAN_EN	189	VDD	88
JTAG_TRST	162	PCI_STOPn	40	SB_D0	99	SE_ACK	159	VDD	232
PCI_AD0	70	PCI_TRDYn	38	SB_D1	100	SE_CLK	160	VDD2	107
PCI_AD1	69	PLL_AGND	168	SB_D2	101	SE_DI	161	VDD2	137
PCI_AD2	67	PLL_LP2	169	SB_D3	102	SYS_BOOT0	173	VDD2	17
PCI_AD3	66	PLL_VDD	167	SB_D4	103	SYS_BOOT1	174	VDD2	164
PCI_AD4	65	PLL_VSS	170	SB_D5	108	SYS_CLK	166	VDD2	195
PCI_AD5	64	PLL_IDDTn	172	SB_D6	109	SYS_CLK_PCI	165	VDD	217
PCI_AD6	63	RX_ADDR0	222	SB_D7	110	SYS_CPCOND	175	VDD2	32
PCI_AD7	62	RX_ADDR1	223	SB_D8	111	SYS_INTn0	176	VDD2	48
PCI_AD8	60	RX_ADDR2	224	SB_D9	112	SYS_INTn1	177	VDD2	78
PCI_AD9	58	RX_ADDR3	225	SB_D10	113	SYS_NMIn	178	VSS	4
PCI_AD10	55	RX_ADDR4	226	SB_D11	114	SYS_OE	179	VSS	104
PCI_AD11	54	RX_CLAV0	221	SB_D12	115	SYS_PLL	171	VSS	120
PCI_AD12	53	RX_CLAV1	220	SB_D13	116	SYS_PSTALLn	182	VSS	134
PCI_AD13	52	RX_CLAV2	219	SB_D14	117	TEST_EN	188	VSS	14
PCI_AD14	51	RX_CLAV3	218	SB_D15	118	TM_CLK	183	VSS	150
PCI_AD15	50	RX_CLK	215	SB_D16	119	TX_ADDR0	190	VSS	180
PCI_AD16	33	RX_DATA0	227	SB_D17	122	TX_ADDR1	191	VSS	213
PCI_AD17	28	RX_DATA1	228	SB_D18	123	TX_ADDR2	192	VSS	29
PCI_AD18	27	RX_DATA2	229	SB_D19	124	TX_ADDR3	193	VSS	45
PCI_AD19	26	RX_DATA3	230	SB_D20	125	TX_ADDR4	196	VSS	56
PCI_AD20	24	RX_DATA4	231	SB_D21	126	TX_CLAV0	208	VSS	75
PCI_AD21	23	RX_DATA5	234	SB_D22	127	TX_CLAV1	209	VSS	90
PCI_AD22	22	RX_DATA6	235	SB_D23	128	TX_CLAV2	210	VSS	233
PCI_AD23	21	RX_DATA7	236	SB_D24	129	TX_CLAV3	211	VSS2	163
PCI_AD24	18	RX_ENBn	238	SB_D25	130	TX_CLK	214	VSS2	194
PCI_AD25	15	RX_PRTY	237	SB_D26	131	TX_DATA0	197	VSS2	216
PCI_AD26	12	RX_SOC	239	SB_D27	132	TX_DATA1	198	VSS2	31
PCI_AD27	11	SB_A2	71	SB_D28	133	TX_DATA2	199	VSS2	47
PCI_AD28	10	SB_A3	72	SB_D29	138	TX_DATA3	200	VSS2	77
PCI_AD29	9	SB_A4	73	SB_D30	139	TX_DATA4	201	VSS2	106
PCI_AD30	8	SB_A5	79	SB_D31	140	TX_DATA5	202	VSS2	136
PCI_AD31	7	SB_A6	80	SB_GNTn	156	TX_DATA6	203	VSS2	16
PCI_CBEn0	61	SB_A7	81	SB_OEn0	145	TX_DATA7	204	+5V	2
PCI_CBEn1	49	SB_A8	82	SB_OEn1	146	TX_ENBn	206	+5V	13
PCI_CBEn2	34	SB_A9	83	SB_OEn2	147	TX_PRTY	205	+5V	25
PCI_CBEn3	19	SB_A10	84	SB_OEn3	148	TX_SOC	207	+5V	36
PCI_CLK	74	SB_A11	85	SB_PCSn0	149	VDD	3	+5V	44
PCI_DEVSELn	39	SB_A12	86	SB_PCSn1	152	VDD	105	+5V	59
PCI_FRAMEn	35	SB_A13	87	SB_PCSn2	153	VDD	121	+5V	68
PCI_GNTn	5	SB_A14	91	SB_PCSn3	154	VDD	135		
PCI_IDSEL	20	SB_A15	92	SB_PCSn4	155	VDD	151		
PCI_INTn	240	SB_A16	93	SB_RDYn	157	VDD	181		
PCI_IRDYn	37	SB_A17	94	SB_REQn	158	VDD	212		
PCI_PAR	43	SB_A18	95	SB_WEn0	141	VDD	30		

Figure 13.3 L64364 240 Pin PQUAD



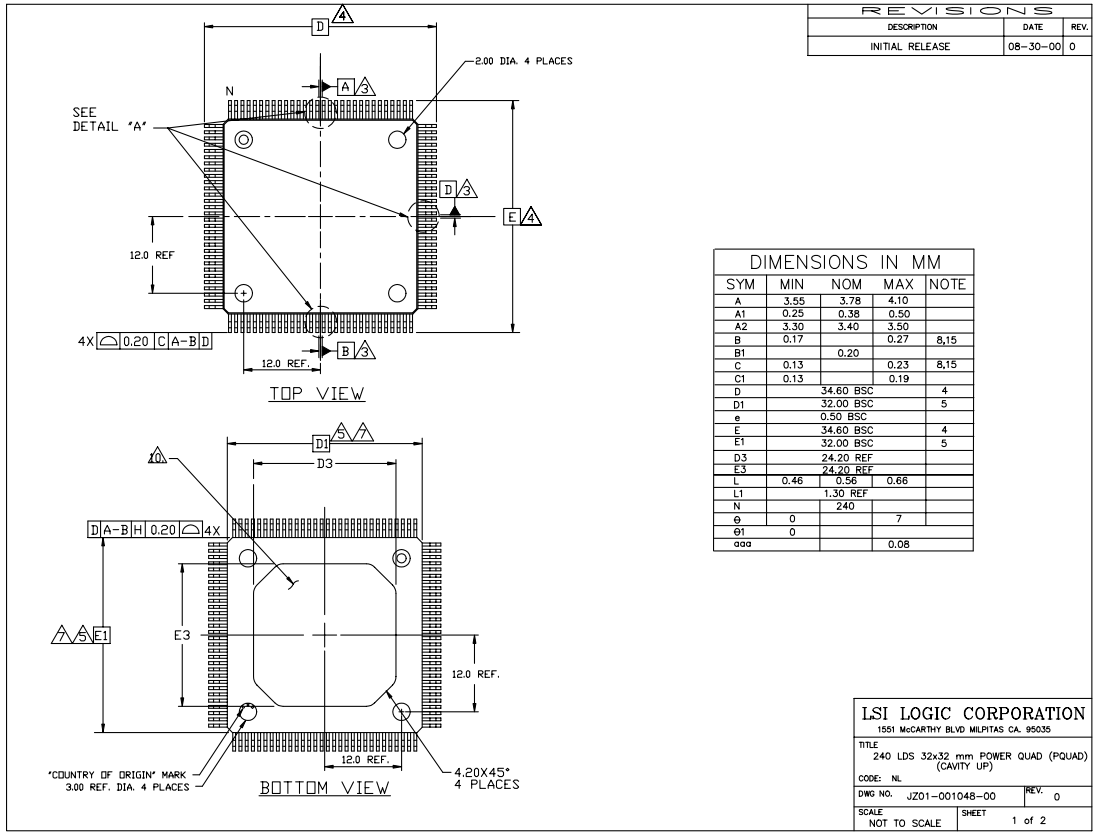
13.4 Package Information

The L64364 is packaged in a 240-pin PQUAD. Electrical and thermal characteristics for the 240-pin PQUAD are summarized in Table 13.9 and its mechanical dimensions are shown in Figure 13.4.

Table 13.9 PQUAD Electrical and Thermal Data

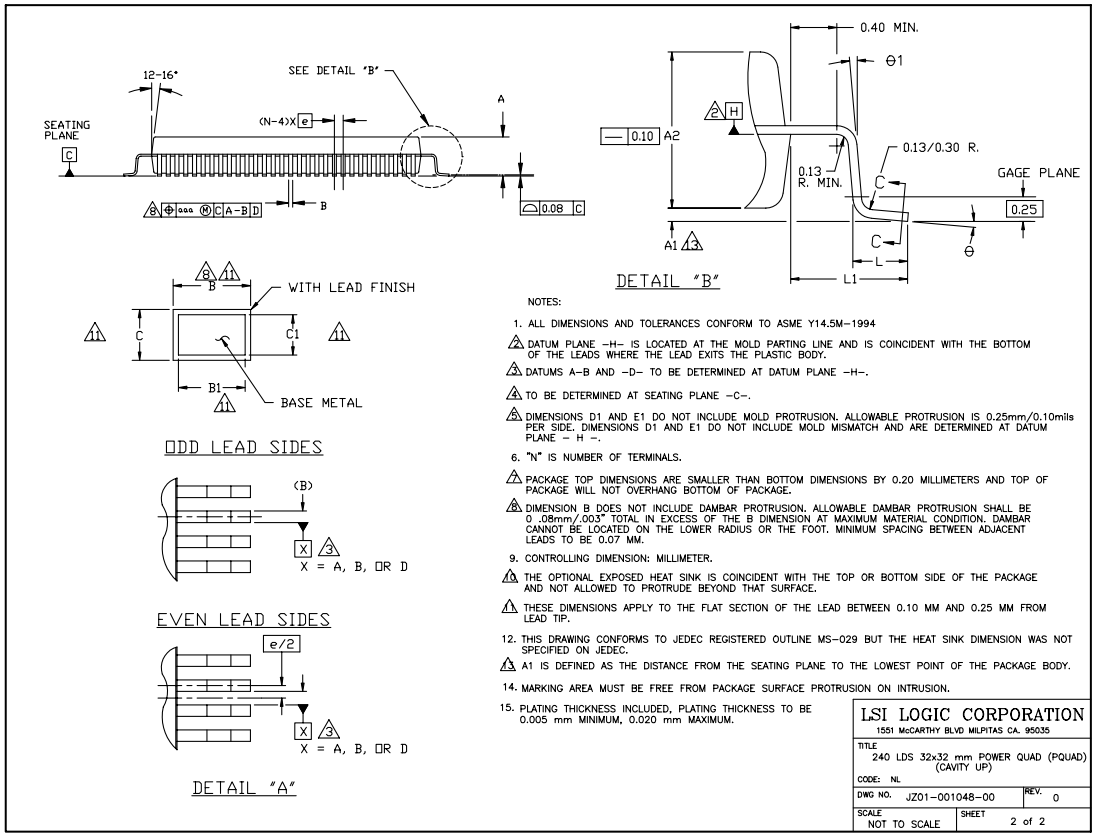
Symbol	Property	Condition	Min	Typ	Max	Units
	Lead Inductance		12.52	–	18.48	nH
	Lead Resistance		127	–	164	MOhms
	Lead Capacitance		1.48	–	2.28	pF
Θ_{ja}	Thermal Resistance	Air flow = 0 LFPM	–	–	12.5	˚C/W
Θ_{ja}	Thermal Resistance	Air flow = 200 LFPM	–	–	9.7	˚C/W
Θ_{ja}	Thermal Resistance	Air flow = 500 LFPM	–	–	8.5	˚C/W

Figure 13.4 240-pin PQUAD (NL) Mechanical Drawing (Sheet 1 of 2)



Important: **This drawing may not be the latest version. For board layout and manufacturing, obtain the most recent engineering drawings from your LSI Logic marketing representative by requesting the outline drawing for package code NL.**

Figure 13.4 240-pin PQuad (NL) Mechanical Drawing (Sheet 2 of 2)



Important: **This drawing may not be the latest version. For board layout and manufacturing, obtain the most recent engineering drawings from your LSI Logic marketing representative by requesting the outline drawing for package code NL.**

Appendix A
Register Summary

CW4011 Data Manipulation Registers				
Name	**Size**	**R/W**	**Description**	**Page**
Rotate	32	R/W	Contains a 5-bit Shift Count used by SELSL and SELSR.	4-49
Circular Mask	32	R/W	Contains a value used in Load and Store address calculations.	4-49

CW4011 Exception Handling Registers				
Name[1]	**Size**	**R/W**	**Description**	**Page**
Debug Control & Status	32	R/W	Contains enable and status bits.	4-63
Count	32	R/W	Functions as a timer.	4-64
Compare	32	R/W	When Timer equals Compare value, generates interrupt.	4-65
Status	32	R/W	Processor status.	4-65, 4-68
Cause	32	R/W	Most recent exception cause.	4-71
EPC	32	R/W	Post exception Program Counter restart address.	4-73
PRID	32	R	Processor implementation and revision number.	4-73
CCC	32	R/W	Cache configuration and control.	4-74
LLAdr	32	R/W	Most recent Load Linked read physical address.	4-77
BPC	32	R/W	Contains program counter breakpoint.	4-78
BDA	32	R/W	Contains virtual data address breakpoint.	4-78
BPCM	32	R/W	Masks BPC bits.	4-78
BDAM	32	R/W	Masks BDA bits.	4-79
Error EPC	32	R/W	Stores PC.	4-79

1. Accessed using MIPS `mtc0` and `mfc0` instructions.

EDMA Registers, 0xB800.0000					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
EDMA_TxCompl	0x00	32	R	Read Transmit Completion Queue.	5-49
EDMA_TxConNum	0x04	32	R/W	Connection Number for the TxCell command.	5-49
EDMA_TxCell	0x08	32	R/W	Issue a TxCell command.	5-49
EDMA_TxConAct	0x10	32	R	Current active ConNum processed by TxCell Processor.	5-49
EDMA_TxComplB	0x14	32	R	Auxiliary Transmit Completion Queue.	5-49

EDMA Registers, 0xB800.0000 (Cont.)					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
EDMA_TxConClose	0x18	32	W	Transmit Connection Close command.	5-49
EDMA_AAL5Pad	0x1F	8	R/W	AAL5 pad byte.	5-49
EDMA_RxCompl	0x40	32	R	Read Receive Completion Queue.	5-49
EDMA_RxConNum	0x44	32	R/W	Connection Number for the RxCell command.	5-50
EDMA_RxCell	0x48	32	R/W	Issue an RxCell command.	5-50
EDMA_RxConAct	0x50	32	R	Current active ConNum processed by RxCell Processor.	5-50
EDMA_RxComplB	0x54	32	R	Auxiliary Receive Completion Queue.	5-50
EDMA_RxConClose	0x58	32	W	Receive Connection Close command.	5-50
EDMA_Buff	0x80	32	R/W	Issue a Buff command.	5-50
EDMA_BuffCompl	0x88	32	R	Read Buff Completion Queue.	5-50
EDMA_BuffComplB	0x8C	32	R	Auxiliary Buff Completion Queue.	5-50
EDMA_BuffConAct	0x90	32	R	Current active ConNum processed by Buff Processor.	5-50
EDMA_LBuff0	0x94	16	R/W	Head of Large Free Buffer list 0.	5-50
EDMA_SBuff0	0x96	16	R/W	Head of Small Free Buffer list 0.	5-50
EDMA_LBuff1	0x98	16	R/W	Head of Large Free Buffer list 1.	5-50
EDMA_SBuff1	0x9A	16	R/W	Head of Small Free Buffer list 1.	5-50
EDMA_MoveSrc	0xA0	32	R/W	Program the source address for a move command.	5-50
EDMA_MoveDst	0xA4	32	R/W	Program the destination address for a move command.	5-50
EDMA_MoveCount	0xA8	32	R/W	Program the byte count and issue a move command.	5-50
EDMA_MoveCount2	0xAC	32	R/W	Program the byte count and issue move command using 32-bit addressing.	5-31
EDMA_LBuff2	0xB0	16	R/W	Head of Large Free Buffer list 2.	5-50
EDMA_SBuff2	0xB2	16	R/W	Head of Small Free Buffer list 2.	5-50
EDMA_LBuff3	0xB4	16	R/W	Head of Large Free Buffer list 3.	5-50
EDMA_SBuff3	0xB6	16	R/W	Head of Small Free Buffer list 3.	5-50
EDMA_LBuff4	0xB8	16	R/W	Head of Large Free Buffer list 4.	5-50
EDMA_SBuff4	0xBA	16	R/W	Head of Small Free Buffer list 4.	5-50
EDMA_LBuff5	0xBC	16	R/W	Head of Large Free Buffer list 5.	5-50
EDMA_SBuff5	0xBE	16	R/W	Head of Small Free Buffer list 5.	5-51
EDMA_Ctrl	0xC0	16	R/W	EDMA control bits.	5-51
EDMA_Status	0xC4	16	R	Check the EDMA status.	5-33
EDMA_LBuffSize	0xC8	16	R/W	Size of large buffers in bytes.	5-51
EDMA_SBuffSize	0xCA	16	R/W	Size of small buffers in bytes.	5-51
EDMA_VCD_Base	0xCC	32	R/W	Base address of the VC Descriptor Table.	5-51
EDMA_BFD_LBase	0xD0	32	R/W	Local Base address of the Buffer Descriptor Table.	5-51
EDMA_BFD_FBase	0xD4	32	R/W	Far Base address of the Buffer Descriptor Table.	5-51
EDMA_ErrMask	0xDC	16	R/W	Error Mask register.	5-51
EDMA_BusErr	0xE3	8	R	Address and Bus error register.	5-51

ACI Registers, 0xB800.0100					
Name	**Offset**	**Size**	**R/W**	**Description**	**Pages**
ACI_Ctrl	0x00	16	R/W	ACI Control field.	6-9, 6-10
ACI_FreeList	0x02	16	R/W	Beginning of free cell list.	6-9, 6-12
ACI_TxTimer	0x04	8	R/W	Transmit time-out.	6-9, 6-13
ACI_TxSize	0x05	8	R/W	Maximum number of cells in Tx FIFO.	6-9, 6-14
ACI_TxLimit	0x06	8	R/W	Number of cells in Tx FIFO to generate an interrupt.	6-9, 6-14
ACI_RxLimit	0x07	8	R/W	Number of cells in Rx FIFO to generate an interrupt.	6-9, 6-14
ACI_RxMask	0x08	32	R/W	Receive polling mask.	6-9, 6-14
ACI_Free	0x0C	32	R/W	Get or return a free cell location.	6-9, 6-14
ACI_RxRead	0x10	32	R	Get cell from Rx FIFO.	6-9, 6-15
ACI_TxWrite	0x14	32	W	Put cell in Tx FIFO.	6-9, 6-15
ACI_RxCells	0x18	8	R	Number of cells in the Rx FIFO.	6-9, 6-16
ACI_TxCells	0x1A	8	R	Number of cells in the Tx FIFO.	6-9, 6-16
ACI_Error	0x1C	32	R	Get a cell from the Error FIFO.	6-9, 6-16
ACI_RxSize	0x20	8	R/W	Maximum number of cells in Receive FIFO.	6-9, 6-17
ACI_BadHEC	0x26	16	R/W	Bad HEC register.	6-9, 6-17
ACI_ClearBytes	0x2B	8	R/W	ACI will use this for PAD byte.	6-9, 6-18
ACI_FreeCount	0x2F	8	R/W	Count of Free Cells.	6-9, 6-18

Scheduler Unit Registers, 0xB800.0200					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
SCD_Ctrl	0x00	32	R/W	Control register.	7-15
SCD_CalSize0	0x06	16	R/W	Size of the Calendar Table 0.	7-14
SCD_Now	0x0A	16	R/W	Current cell slot pointer.	7-14
SCD_Serv	0x0C	32	R	Execute `service` command.	7-14
SCD_Sched	0x10	32	W	Execute `schedule` command.	7-6
SCD_Tic	0x18	32	RW	Execute `tic` command.	7-14
SCD_CalSwitch	0x23	8	R/W	Execute `Cal_Switch` command.	7-11
SCD_CalBase1	0x28	32	R/W	Base of Calendar Table 1.	7-14
SCD_CalBase2	0x2C	32	R/W	Base of Calendar Table 2.	7-14
SCD_CalBase3	0x30	32	R/W	Base of Calendar Table 3.	7-14
SCD_CalSize1	0x36	16	R/W	Size of the Calendar Table 1.	7-14
SCD_CalSize2	0x3A	16	R/W	Size of the Calendar Table 2.	7-14
SCD_CalSize3	0x3E	16	R/W	Size of the Calendar Table 3.	7-14
SCD_HeadSel	0x43	8	R/W	Head Insertion selection.	7-7
SCD_Err	0x47	8	R	Error register.	7-17
SCD_Class0	0x48	32	R	Head & Tail of Priority Class 0.	7-14
SCD_Class1	0x4C	32	R	Head & Tail of Priority Class 1.	7-14
SCD_Class2	0x50	32	R	Head & Tail of Priority Class 2.	7-14
SCD_Class3	0x54	32	R	Head & Tail of Priority Class 3.	7-14
SCD_Class4	0x58	32	R	Head & Tail of Priority Class 4.	7-14
SCD_Class5	0x5C	32	R	Head & Tail of Priority Class 5.	7-14

Timer Unit Registers, 0xB800.0280					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
TM_TimeStamp	0x00	32	R/W	Time Stamp counter.	8-2
TM_Timer1	0x04	8	R/W	Timer value.	8-2
TM_TimerInit1	0x06	8	R/W	Timer initialization value.	8-2
TM_Timer2	0x08	8	R/W	Timer value.	8-2
TM_TimerInit2	0x0A	8	R/W	Timer initialization value.	8-2
TM_Timer3	0x0C	8	R/W	Timer value.	8-2
TM_TimerInit3	0x0E	8	R/W	Timer initialization value.	8-2
TM_Timer4	0x10	8	R/W	Timer value.	8-2
TM_TimerInit4	0x12	8	R/W	Timer initialization value.	8-2
TM_Timer5	0x14	8	R/W	Timer value.	8-2
TM_TimerInit5	0x16	8	R/W	Timer initialization value.	8-2
TM_Timer6	0x18	8	R/W	Timer value.	8-2
TM_TimerInit6	0x1A	8	R/W	Timer initialization value.	8-2
TM_Timer7	0x1C	8	R/W	Timer value.	8-2
TM_TimerInit7	0x1E	8	R/W	Timer initialization value.	8-2
TM_Enable	0x20	6	R/W	Time-out enable.	8-2
TM_Clear	0x24	6	W	Time-out clear.	8-2
TM_ClockSel	0x28	32	R/W	Timer clock selection.	8-3
TM_ClockSel2	0x2C	8	R/W	Timer clock selection 2.	8-4
TM_Timer8	0x30	8	R/W	Timer Value.	8-2
TM_TimerInit8	0x32	8	R/W	Timer initialization value.	8-2

APU Registers, 0xB800.0300					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
APU_AddrMap	0x00	32	R/W	Holds MSBs to extend external addresses.	4-99
APU_SCbus_Watchdog	0x06	16	R/W	Timer for APU SC Bus transactions.	4-113
APU_SRL	0x08	32	R	APU Serial register.	4-119
APU_VIntEnable	0x0E	16	R/W	Enables/masks vectored interrupts.	4-106
APU_VIntBase	0x10	32	R/W	Holds part of EVI handler routine address.	4-107
APU_Status	0x14	32	R	Holds status of real-time events.	4-108
APU_OCAbus_Watchdog	0x1A	16	R/W	Timer for APU OCA Bus transactions.	4-112
APU_Priority	0x1F	8	R/W	Priority raise register for SC Bus transactions.	4-114
APU_Error	0x20	32	R/W	Error register for SC Bus time-out and nonvectored interrupt status.	4-115
OCA_Error	0x80	32	R/W	Error register for OCAbus time-outs.	4-117

Primary Port Controller Registers, 0xB800.0400					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
PP_Ctrl	0x03	8	R/W	Specifies burst size limit	9-24
PP_RxMbx	0x04	32	R	Primary Port Rx Mailbox.	9-31
PP_TxMbx	0x08	32	W	Primary Port Tx Mailbox.	9-31
PP_SlavePFtch	0x10	32	R/W	Primary Port Slave Prefetch register.	9-25
PP_Err	0x20	32	R/W	Error register.	9-27
PP_ErrAddr	0x24	32	R	Error Address register.	9-29

SBC Control Registers, 0xB800.0800					
Name	**Offset**	**Size**	**R/W**	**Description**	**Page**
SP_Ctrl	0x00	32	R/W	Enables local memory pages and the number of wait states.	10-4
SP_SDRAM	0x04	32	R/W	Various SDRAM control functions.	10-16
SP_Refresh	0x08	32	Mixed	Refresh interval count and counter.	10-20
SB Clock Control	0x10	32	R/W	SB_DCLK and SB_CLKO timing adjustments.	10-7
SP_Err	0x20	32	Mixed	Error register.	10-41
SP_ErrAddr	0x24	32	R	Error Address register.	10-42

PCI Configuration Registers (See Section 9.2, "PCI Configuration Space Registers.")					
Name	**Address**	**Size**	**R/W**	**Description**	**Page**
Vendor ID	0x00	16	Read	LSI Logic vendor ID number.	9-6
Device ID	0x02	16	Read	PCI device identification number.	9-6
Command	0x04	16	Mixed	PCI control.	9-7
Status	0x06	16	Mixed	Status of operation and data.	9-8
Revision ID	0x08	8	Read	L64364 revision number.	9-10
Class Code	0x09	24	Read	ATM controller class code.	9-10
Cache Line Size	0x0C	8	R/W	Cache line size.	9-11
Latency Timer	0x0D	8	R/W	Length of PCI bus ownership after PCI_GNTn deasserted.	9-11
Header Type	0x0E	8	Read	PCI header type.	9-12
Base Address Register 1	0x10	32	Mixed	PCI base address and parameters for Cell buffer, mailbox, and XPP_Ctrl register.	9-12
Base Address Register 2	0x14	32	Mixed	PCI base address and parameters for local memory.	9-13
Sub Vend ID	0x2C	16	R/W	Subsystem Vendor ID Number.	9-15
Subsystem ID	0x2E	16	R/W	Subsystem ID Number.	9-15
Interrupt Line	0x3C	8	R/W	PCI interrupt line number.	9-16
Interrupt Pin	0x3D	8	Read	PCI interrupt pin.	9-16

PCI Configuration Registers (See Section 9.2, "PCI Configuration Space Registers.") (Cont.)					
Name	**Address**	**Size**	**R/W**	**Description**	**Page**
Minimum Grant	0x3E	8	Read	Value for minimum PCI grant duration is hardwired to 0x00.	9-17
Maximum Latency	0x3F	8	Read	Maximum PCI bus latency is hardwired to 0x00.	9-17
TRDY Timer	0x40	8	R/W	IRDY to TRDY timing.	9-18
RETRY Timer	0x41	8	R/W	Maximum number of PCI master access retries.	9-18
XPP_Ctrl	Note[1]	8	Mixed	Various control functions.	9-22
Rx/Tx MailBox	Note[2]	32	R/W	Write = Rx, Read = Tx	9-30

1. Base Address 1 + 0x4010
2. Base Address 1 + 0x4000

Appendix B
The ATM Cell

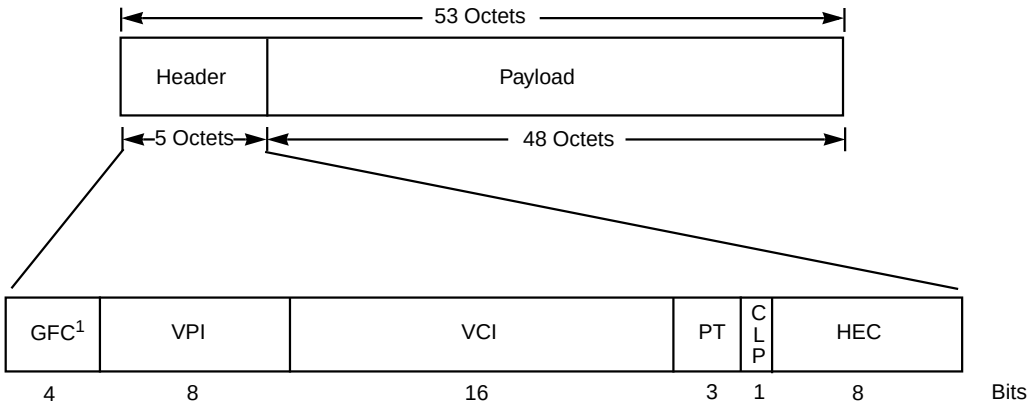
This appendix contains information about the structure of the ATM cell header and the AAL5 mode CS-PDU trailer. It contains the following sections:

- Section B.1, “ATM Cell Structure,” page B-1
- Section B.2, “The AAL5 Trailer,” page B-3

B.1 ATM Cell Structure

The unit of transmission across ATM networks is called a cell. A cell (see Figure B.1) consists of a 48-octet (byte) payload of user data and a 5-octet header. The header fields for the User Network Interface (UNI) are described in Figure B.1.

Figure B.1 The ATM Cell Layout at the UNI



1. At the Network Node Interface (NNI), GFC is not used and the VPI is extended into this field to become 12-bits long.

GFC **Generic Flow Control** **4 Bits**

This field permits a non-ATM unit, such as a multiplexer, to control the rate of data flow between it and an ATM terminal. At the Network Node Interface (NNI), these bits are used for an extended VPI.

VPI **Virtual Path Indicator** **8 Bits**

A transmission path between two ATM units which, in turn, may contain virtual channels. An intermediate ATM unit (such as a switch) changes the VPI of incoming cells to denote the output channel (port) to which they belong. A cell can be assigned to one of 256 virtual paths.

At the Network Node Interface (NNI), the VPI is extended to 12-bits starting at the beginning of the header. This provides 4096 virtual paths.

VCI **Virtual Channel Indicator** **16 Bits**

An intermediate ATM unit changes the VCIs of incoming cells to group them for further switching to a common path. The combination of VPIs and VCIs defines the exact route of cells through an ATM network. Each virtual path can be divided into 2^{16} virtual channels.

PT **Payload Type** **3 Bits**

The PT field code is used to define the payload as user, signalling, or maintenance data. Encoding of the PT field is shown below:

Code	Definition
0b000	MSB = 0 signifies a user data cell. The middle bit is the Explicit Forward Congestion Indicator (EFCI). The LSB is the AAL_indicate bit. In AAL5 mode, it is 0 for all cells except the last one in a CS-PDU. In this mode the last cell contains a CS-PDU trailer.
0b001	
0b010	
0b011	
0b100	Operations Administration and Maintenance (OAM) segment cell.
0b101	OAM end-to-end cell.
0b110	Resource management cell.
ob111	Reserved for future use.

CLP **Cell Loss Priority** **1 Bit**
This bit indicates the priority the cell has over other cells in a congested traffic network. If necessary, cells coded 1 will be discarded first.

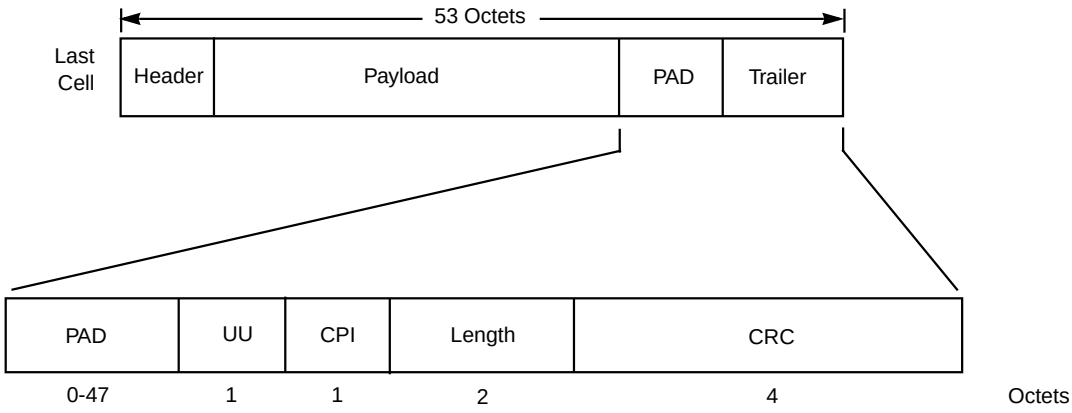
HEC **Header Error Check** **8 Bits**
Since the information in the header is critical, the HEC field is used to check for and correct errors. Payload error checking is left to higher layer protocols.

B.2 The AAL5 Trailer

In AAL5 mode, a pad and a trailer are added to the CS-PDU. The trailer (see Figure B.2) is eight octets wide. Its fields are defined following the figure. The pad is inserted to make the CS-PDU a multiple of 48 octets so it can be evenly divided into cell payloads.

The AAL_indicate bit in the PT field of the cell headers is 0 and changes to 1 in the last cell of the CS-PDU to indicate that the cell contains the trailer.

Figure B.2 The AAL5 Trailer Layout



UU **User-to-User Indicator** **1 Octet**
For user-to-user information. Transparent to the ATM network.

CPI	**Common Part Indicator** This field is set to 0x00 per the current ATM specifications.	**1 Octet**
Length	**CS-PDU Length** Identifies the length of the CS-PDU payload and determines the size of the pad.	**2 Octets**
CRC	**Cyclic Redundancy Check** CRC-32 for detecting errors in the CS-PDU payload.	**4 Octets**

Appendix C
Glossary of Abbreviations

This appendix defines some of the abbreviations used in this manual.

Abbreviation	Definition
AAL1, 2, 3/4, 5	ATM Adaptation Layer 1, 2, 3/4, 5
ABR	Available Bit Rate
ACI	ATM Cell Interface
ALU	Arithmetic Logic Unit
APU	ATM Processing Unit
ASSP	Application Specific Standard Product
ATM	Asynchronous Transfer Mode
BFD	Buffer Descriptor
BIU	Bus Interface Unit
BOM	Beginning of Message
CAS	Column Address Select
CBM	Cell Buffer Memory
CBR	Constant Bit Rate (ATM)
CBR	Column Before Row (DRAM Refresh)
CLP	Call Loss Priority
CP0	Coprocessor 0
CRC	Cyclic Redundancy Check
CS-PDU	Convergence Sublayer - Protocol Data Unit
EDMA	Enhanced Direct Memory Access
EFCI	Explicit Forward Congestion Indicator
EOM	End of Message
EPC	Exception Program Counter
EVI	External Vectored Interrupt

Abbreviation	Definition
FIFO	First In First Out
HEC	Header Error Check
ISA	Industry Standard Architecture
ISU	Instruction Schedule Unit
JTAG	IEEE 1149.1 boundary scan standard
LRU	Least Recently Used
LSB	Least Significant Bit/Byte
LSU	Load/Store/Add Unit
MIPS	Millions of Instructions Per Second
MMU	Memory Management Unit
MSB	Most Significant Bit/Byte
NMI	Nonmaskable Interrupt
NNI	Network Node Interface
NOP	No Operation
PC	Program Counter
PCI	Peripheral Component Interconnect
PDU	Protocol Data Unit
PHY	Physical Layer or Device
PLL	Phase-Locked Loop
QoS	Quality of Service
RAS	Row Address Select
RISC	Reduced Instruction Set Computer
SAR	Segmentation and Reassembly
TLB	Translation Lookaside Buffer
UBR	Unspecified Bit Rate
UNI	User Network Interface
UU	User to User
VBR	Variable Bit Rate

Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at the number shown.

If appropriate, please also fax copies of any marked-up pages from this document.

Important: Please include your name, phone number, fax number, and company address so that we may contact you directly for clarification or additional information.

Thank you for your help in improving the quality of our documents.

Reader's Comments

Fax your comments to: LSI Logic Corporation
Technical Publications
M/S E-198
Fax: 408.433.4333

Please tell us how you rate this document: *L64364 ATMizer II+ ATM-SAR Chip Technical Manual.* Place a check mark in the appropriate blank for each category.

	Excellent	Good	Average	Fair	Poor
Completeness of information	____	____	____	____	____
Clarity of information	____	____	____	____	____
Ease of finding information	____	____	____	____	____
Technical content	____	____	____	____	____
Usefulness of examples and illustrations	____	____	____	____	____
Overall manual	____	____	____	____	____

What could we do to improve this document?

_____

_____

_____

_____

If you found errors in this document, please specify the error and page number. If appropriate, please fax a marked-up copy of the page(s).

_____

_____

_____

Please complete the information below so that we may contact you directly for clarification or additional information.

Name _____ Date _____

Telephone _____ Fax _____

Title _____

Department _____ Mail Stop _____

Company Name _____

Street _____

City, State, Zip _____

U.S. Distributors by State

A. E. Avnet Electronics
http://www.hh.avnet.com
B. M. Bell Microproducts, Inc. (for HAB's)
http://www.bellmicro.com
I. E. Insight Electronics
http://www.insight-electronics.com
W. E. Wyle Electronics
http://www.wyle.com

Alabama
Daphne
I. E. Tel: 334.626.6190
Huntsville
A. E. Tel: 256.837.8700
B. M. Tel: 256.705.3559
I. E. Tel: 256.830.1222
W. E. Tel: 800.964.9953

Alaska
A. E. Tel: 800.332.8638

Arizona
Phoenix
A. E. Tel: 480.736.7000
B. M. Tel: 602.267.9551
W. E. Tel: 800.528.4040
Tempe
I. E. Tel: 480.829.1800
Tucson
A. E. Tel: 520.742.0515

Arkansas
W. E. Tel: 972.235.9953

California
Agoura Hills
B. M. Tel: 818.865.0266
Granite Bay
B. M. Tel: 916.523.7047
Irvine
A. E. Tel: 949.789.4100
B. M. Tel: 949.470.2900
I. E. Tel: 949.727.3291
W. E. Tel: 800.626.9953
Los Angeles
A. E. Tel: 818.594.0404
W. E. Tel: 800.288.9953
Sacramento
A. E. Tel: 916.632.4500
W. E. Tel: 800.627.9953
San Diego
A. E. Tel: 858.385.7500
B. M. Tel: 858.597.3010
I. E. Tel: 800.677.6011
W. E. Tel: 800.829.9953
San Jose
A. E. Tel: 408.435.3500
B. M. Tel: 408.436.0881
I. E. Tel: 408.952.7000
Santa Clara
W. E. Tel: 800.866.9953
Woodland Hills
A. E. Tel: 818.594.0404
Westlake Village
I. E. Tel: 818.707.2101

Colorado
Denver
A. E. Tel: 303.790.1662
B. M. Tel: 303.846.3065
W. E. Tel: 800.933.9953
Englewood
I. E. Tel: 303.649.1800
Idaho Springs
B. M. Tel: 303.567.0703

Connecticut
Cheshire
A. E. Tel: 203.271.5700
I. E. Tel: 203.272.5843
Wallingford
W. E. Tel: 800.605.9953

Delaware
North/South
A. E. Tel: 800.526.4812
Tel: 800.638.5988
B. M. Tel: 302.328.8968
W. E. Tel: 856.439.9110

Florida
Altamonte Springs
B. M. Tel: 407.682.1199
I. E. Tel: 407.834.6310
Boca Raton
I. E. Tel: 561.997.2540
Bonita Springs
B. M. Tel: 941.498.6011
Clearwater
I. E. Tel: 727.524.8850
Fort Lauderdale
A. E. Tel: 954.484.5482
W. E. Tel: 800.568.9953
Miami
B. M. Tel: 305.477.6406
Orlando
A. E. Tel: 407.657.3300
W. E. Tel: 407.740.7450
Tampa
W. E. Tel: 800.395.9953
St. Petersburg
A. E. Tel: 727.507.5000

Georgia
Atlanta
A. E. Tel: 770.623.4400
B. M. Tel: 770.980.4922
W. E. Tel: 800.876.9953
Duluth
I. E. Tel: 678.584.0812

Hawaii
A. E. Tel: 800.851.2282

Idaho
A. E. Tel: 801.365.3800
W. E. Tel: 801.974.9953

Illinois
North/South
A. E. Tel: 847.797.7300
Tel: 314.291.5350
Chicago
B. M. Tel: 847.413.8530
W. E. Tel: 800.853.9953
Schaumburg
I. E. Tel: 847.885.9700

Indiana
Fort Wayne
I. E. Tel: 219.436.4250
W. E. Tel: 888.358.9953
Indianapolis
A. E. Tel: 317.575.3500

Iowa
W. E. Tel: 612.853.2280
Cedar Rapids
A. E. Tel: 319.393.0033

Kansas
W. E. Tel: 303.457.9953
Kansas City
A. E. Tel: 913.663.7900
Lenexa
I. E. Tel: 913.492.0408

Kentucky
W. E. Tel: 937.436.9953
Central/Northern/ Western
A. E. Tel: 800.984.9503
Tel: 800.767.0329
Tel: 800.829.0146

Louisiana
W. E. Tel: 713.854.9953
North/South
A. E. Tel: 800.231.0253
Tel: 800.231.5775

Maine
A. E. Tel: 800.272.9255
W. E. Tel: 781.271.9953

Maryland
Baltimore
A. E. Tel: 410.720.3400
W. E. Tel: 800.863.9953
Columbia
B. M. Tel: 800.673.7461
I. E. Tel: 410.381.3131

Massachusetts
Boston
A. E. Tel: 978.532.9808
W. E. Tel: 800.444.9953
Burlington
I. E. Tel: 781.270.9400
Marlborough
B. M. Tel: 800.673.7459
Woburn
B. M. Tel: 800.552.4305

Michigan
Brighton
I. E. Tel: 810.229.7710
Detroit
A. E. Tel: 734.416.5800
W. E. Tel: 888.318.9953
Clarkston
B. M. Tel: 877.922.9363

Minnesota
Champlin
B. M. Tel: 800.557.2566
Eden Prairie
B. M. Tel: 800.255.1469
Minneapolis
A. E. Tel: 612.346.3000
W. E. Tel: 800.860.9953
St. Louis Park
I. E. Tel: 612.525.9999

Mississippi
A. E. Tel: 800.633.2918
W. E. Tel: 256.830.1119

Missouri
W. E. Tel: 630.620.0969
St. Louis
A. E. Tel: 314.291.5350
I. E. Tel: 314.872.2182

Montana
A. E. Tel: 800.526.1741
W. E. Tel: 801.974.9953

Nebraska
A. E. Tel: 800.332.4375
W. E. Tel: 303.457.9953

Nevada
Las Vegas
A. E. Tel: 800.528.8471
W. E. Tel: 702.765.7117

New Hampshire
A. E. Tel: 800.272.9255
W. E. Tel: 781.271.9953

New Jersey
North/South
A. E. Tel: 201.515.1641
Tel: 609.222.6400
Mt. Laurel
I. E. Tel: 856.222.9566
Pine Brook
B. M. Tel: 973.244.9668
W. E. Tel: 800.862.9953
Parsippany
I. E. Tel: 973.299.4425
Wayne
W. E. Tel: 973.237.9010

New Mexico
W. E. Tel: 480.804.7000
Albuquerque
A. E. Tel: 505.293.5119

U.S. Distributors by State (Continued)

New York
Hauppauge
I. E. Tel: 516.761.0960
Long Island
A. E. Tel: 516.434.7400
W. E. Tel: 800.861.9953
Rochester
A. E. Tel: 716.475.9130
I. E. Tel: 716.242.7790
W. E. Tel: 800.319.9953
Smithtown
B. M. Tel: 800.543.2008
Syracuse
A. E. Tel: 315.449.4927

North Carolina
Raleigh
A. E. Tel: 919.859.9159
I. E. Tel: 919.873.9922
W. E. Tel: 800.560.9953

North Dakota
A. E. Tel: 800.829.0116
W. E. Tel: 612.853.2280

Ohio
Cleveland
A. E. Tel: 216.498.1100
W. E. Tel: 800.763.9953
Dayton
A. E. Tel: 614.888.3313
I. E. Tel: 937.253.7501
W. E. Tel: 800.575.9953
Strongsville
B. M. Tel: 440.238.0404
Valley View
I. E. Tel: 216.520.4333

Oklahoma
W. E. Tel: 972.235.9953
Tulsa
A. E. Tel: 918.459.6000
I. E. Tel: 918.665.4664

Oregon
Beaverton
B. M. Tel: 503.524.1075
I. E. Tel: 503.644.3300
Portland
A. E. Tel: 503.526.6200
W. E. Tel: 800.879.9953

Pennsylvania
Mercer
I. E. Tel: 412.662.2707
Philadelphia
A. E. Tel: 800.526.4812
B. M. Tel: 877.351.2355
W. E. Tel: 800.871.9953
Pittsburgh
A. E. Tel: 412.281.4150
W. E. Tel: 440.248.9996

Rhode Island
A. E. 800.272.9255
W. E. Tel: 781.271.9953

South Carolina
A. E. Tel: 919.872.0712
W. E. Tel: 919.469.1502

South Dakota
A. E. Tel: 800.829.0116
W. E. Tel: 612.853.2280

Tennessee
W. E. Tel: 256.830.1119
East/West
A. E. Tel: 800.241.8182
Tel: 800.633.2918

Texas
Arlington
B. M. Tel: 817.417.5993
Austin
A. E. Tel: 512.219.3700
B. M. Tel: 512.258.0725
I. E. Tel: 512.719.3090
W. E. Tel: 800.365.9953
Dallas
A. E. Tel: 214.553.4300
B. M. Tel: 972.783.4191
W. E. Tel: 800.955.9953
El Paso
A. E. Tel: 800.526.9238
Houston
A. E. Tel: 713.781.6100
B. M. Tel: 713.917.0663
W. E. Tel: 800.888.9953
Richardson
I. E. Tel: 972.783.0800
Rio Grande Valley
A. E. Tel: 210.412.2047
Stafford
I. E. Tel: 281.277.8200

Utah
Centerville
B. M. Tel: 801.295.3900
Murray
I. E. Tel: 801.288.9001
Salt Lake City
A. E. Tel: 801.365.3800
W. E. Tel: 800.477.9953

Vermont
A. E. Tel: 800.272.9255
W. E. Tel: 716.334.5970

Virginia
A. E. Tel: 800.638.5988
W. E. Tel: 301.604.8488
Haymarket
B. M. Tel: 703.754.3399
Springfield
B. M. Tel: 703.644.9045

Washington
Kirkland
I. E. Tel: 425.820.8100
Maple Valley
B. M. Tel: 206.223.0080
Seattle
A. E. Tel: 425.882.7000
W. E. Tel: 800.248.9953

West Virginia
A. E. Tel: 800.638.5988

Wisconsin
Milwaukee
A. E. Tel: 414.513.1500
W. E. Tel: 800.867.9953
Wauwatosa
I. E. Tel: 414.258.5338

Wyoming
A. E. Tel: 800.332.9326
W. E. Tel: 801.974.9953

Sales Offices and Design Resource Centers

LSI Logic Corporation
Corporate Headquarters
1551 McCarthy Blvd
Milpitas CA 95035
Tel: 408.433.8000
Fax: 408.433.8989

NORTH AMERICA

California
Irvine
18301 Von Karman Ave
Suite 900
Irvine, CA 92612
♦ Tel: 949.809.4600
Fax: 949.809.4444

Pleasanton Design Center
5050 Hopyard Road, 3rd Floor
Suite 300
Pleasanton, CA 94588
Tel: 925.730.8800
Fax: 925.730.8700

San Diego
7585 Ronson Road
Suite 100
San Diego, CA 92111
Tel: 858.467.6981
Fax: 858.496.0548

Silicon Valley
1551 McCarthy Blvd
Sales Office
M/S C-500
Milpitas, CA 95035
♦ Tel: 408.433.8000
Fax: 408.954.3353
Design Center
M/S C-410
Tel: 408.433.8000
Fax: 408.433.7695

Wireless Design Center
11452 El Camino Real
Suite 210
San Diego, CA 92130
Tel: 858.350.5560
Fax: 858.350.0171

Colorado
Boulder
4940 Pearl East Circle
Suite 201
Boulder, CO 80301
♦ Tel: 303.447.3800
Fax: 303.541.0641

Colorado Springs
4420 Arrowswest Drive
Colorado Springs, CO 80907
Tel: 719.533.7000
Fax: 719.533.7020

Fort Collins
2001 Danfield Court
Fort Collins, CO 80525
Tel: 970.223.5100
Fax: 970.206.5549

Florida
Boca Raton
2255 Glades Road
Suite 324A
Boca Raton, FL 33431
Tel: 561.989.3236
Fax: 561.989.3237

Georgia
Alpharetta
2475 North Winds Parkway
Suite 200
Alpharetta, GA 30004
Tel: 770.753.6146
Fax: 770.753.6147

Illinois
Oakbrook Terrace
Two Mid American Plaza
Suite 800
Oakbrook Terrace, IL 60181
Tel: 630.954.2234
Fax: 630.954.2235

Kentucky
Bowling Green
1262 Chestnut Street
Bowling Green, KY 42101
Tel: 270.793.0010
Fax: 270.793.0040

Maryland
Bethesda
6903 Rockledge Drive
Suite 230
Bethesda, MD 20817
Tel: 301.897.5800
Fax: 301.897.8389

Massachusetts
Waltham
200 West Street
Waltham, MA 02451
♦ Tel: 781.890.0180
Fax: 781.890.6158

Burlington - Mint Technology
77 South Bedford Street
Burlington, MA 01803
Tel: 781.685.3800
Fax: 781.685.3801

Minnesota
Minneapolis
8300 Norman Center Drive
Suite 730
Minneapolis, MN 55437
♦ Tel: 612.921.8300
Fax: 612.921.8399

New Jersey
Red Bank
125 Half Mile Road
Suite 200
Red Bank, NJ 07701
Tel: 732.933.2656
Fax: 732.933.2643

Cherry Hill - Mint Technology
215 Longstone Drive
Cherry Hill, NJ 08003
Tel: 856.489.5530
Fax: 856.489.5531

New York
Fairport
550 Willowbrook Office Park
Fairport, NY 14450
Tel: 716.218.0020
Fax: 716.218.9010

North Carolina
Raleigh
Phase II
4601 Six Forks Road
Suite 528
Raleigh, NC 27609
Tel: 919.785.4520
Fax: 919.783.8909

Oregon
Beaverton
15455 NW Greenbrier Parkway
Suite 235
Beaverton, OR 97006
Tel: 503.645.0589
Fax: 503.645.6612

Texas
Austin
9020 Capital of TX Highway North
Building 1
Suite 150
Austin, TX 78759
Tel: 512.388.7294
Fax: 512.388.4171

Plano
500 North Central Expressway
Suite 440
Plano, TX 75074
♦ Tel: 972.244.5000
Fax: 972.244.5001

Houston
20405 State Highway 249
Suite 450
Houston, TX 77070
Tel: 281.379.7800
Fax: 281.379.7818

Canada
Ontario
Ottawa
260 Hearst Way
Suite 400
Kanata, ON K2L 3H1
♦ Tel: 613.592.1263
Fax: 613.592.3253

INTERNATIONAL

France
Paris
LSI Logic S.A.
Immeuble Europa
53 bis Avenue de l'Europe
B.P. 139
78148 Velizy-Villacoublay
Cedex, Paris
♦ Tel: 33.1.34.63.13.13
Fax: 33.1.34.63.13.19

Germany
Munich
LSI Logic GmbH
Orleansstrasse 4
81669 Munich
♦ Tel: 49.89.4.58.33.0
Fax: 49.89.4.58.33.108

Stuttgart
Mittlerer Pfad 4
D-70499 Stuttgart
♦ Tel: 49.711.13.96.90
Fax: 49.711.86.61.428

Italy
Milan
LSI Logic S.P.A.
Centro Direzionale Colleoni Palazzo
Orione Ingresso 1
20041 Agrate Brianza, Milano
♦ Tel: 39.039.687371
Fax: 39.039.6057867

Japan
Tokyo
LSI Logic K.K.
Rivage-Shinagawa Bldg. 14F
4-1-8 Kounan
Minato-ku, Tokyo 108-0075
♦ Tel: 81.3.5463.7821
Fax: 81.3.5463.7820

Osaka
Crystal Tower 14F
1-2-27 Shiromi
Chuo-ku, Osaka 540-6014
♦ Tel: 81.6.947.5281
Fax: 81.6.947.5287

Sales Offices and Design Resource Centers (Continued)

Korea
Seoul
LSI Logic Corporation of Korea Ltd
10th Fl., Haesung 1 Bldg.
942, Daechi-dong,
Kangnam-ku, Seoul, 135-283
Tel: 82.2.528.3400
Fax: 82.2.528.2250

The Netherlands
Eindhoven
LSI Logic Europe Ltd
World Trade Center Eindhoven
Building 'Rijder'
Bogert 26
5612 LZ Eindhoven
Tel: 31.40.265.3580
Fax: 31.40.296.2109

Singapore
Singapore
LSI Logic Pte Ltd
7 Temasek Boulevard
#28-02 Suntec Tower One
Singapore 038987
Tel: 65.334.9061
Fax: 65.334.4749

Sweden
Stockholm
LSI Logic AB
Finlandsgatan 14
164 74 Kista
♦ Tel: 46.8.444.15.00
Fax: 46.8.750.66.47

Taiwan
Taipei
LSI Logic Asia, Inc.
Taiwan Branch
10/F 156 Min Sheng E. Road
Section 3
Taipei, Taiwan R.O.C.
Tel: 886.2.2718.7828
Fax: 886.2.2718.8869

United Kingdom
Bracknell
LSI Logic Europe Ltd
Greenwood House
London Road
Bracknell, Berkshire RG12 2UB
♦ Tel: 44.1344.426544
Fax: 44.1344.481039

♦ Sales Offices with Design Resource Centers

International Distributors

Australia
New South Wales
Reptechnic Pty Ltd
3/36 Bydown Street
Neutral Bay, NSW 2089
♦ Tel: 612.9953.9844
Fax: 612.9953.9683

Belgium
Acal nv/sa
Lozenberg 4
1932 Zaventem
Tel: 32.2.7205983
Fax: 32.2.7251014

China
Beijing
LSI Logic International Services Inc.
Beijing Representative Office
Room 708
Canway Building
66 Nan Li Shi Lu
Xicheng District
Beijing 100045, China
Tel: 86.10.6804.2534 to 38
Fax: 86.10.6804.2521

France
Rungis Cedex
Azzurri Technology France
22 Rue Saarinen
Sillic 274
94578 Rungis Cedex
Tel: 33.1.41806310
Fax: 33.1.41730340

Germany
Haar
EBV Elektronik
Hans-Pinsel Str. 4
D-85540 Haar
Tel: 49.89.4600980
Fax: 49.89.46009840

Munich
Avnet Emg GmbH
Stahlgruberring 12
81829 Munich
Tel: 49.89.45110102
Fax: 49.89.42.27.75

Wuennenberg-Haaren
Peacock AG
Graf-Zepplin-Str 14
D-33181 Wuennenberg-Haaren
Tel: 49.2957.79.1692
Fax: 49.2957.79.9341

Hong Kong
Hong Kong
AVT Industrial Ltd
Unit 608 Tower 1
Cheung Sha Wan Plaza
833 Cheung Sha Wan Road
Kowloon, Hong Kong
Tel: 852.2428.0008
Fax: 852.2401.2105

Serial System (HK) Ltd
2301 Nanyang Plaza
57 Hung To Road, Kwun Tong
Kowloon, Hong Kong
Tel: 852.2995.7538
Fax: 852.2950.0386

India
Bangalore
Spike Technologies India Private Ltd
951, Vijayalakshmi Complex,
2nd Floor, 24th Main,
J P Nagar II Phase,
Bangalore, India 560078
♦ Tel: 91.80.664.5530
Fax: 91.80.664.9748

Israel
Tel Aviv
Eastronics Ltd
11 Rozanis Street
P.O. Box 39300
Tel Aviv 61392
Tel: 972.3.6458777
Fax: 972.3.6458666

Japan
Tokyo
Daito Electron
Sogo Kojimachi No.3 Bldg
1-6 Kojimachi
Chiyoda-ku, Tokyo 102-8730
Tel: 81.3.3264.0326
Fax: 81.3.3261.3984

Global Electronics Corporation
Nichibei Time24 Bldg. 35 Tansu-cho
Shinjuku-ku, Tokyo 162-0833
Tel: 81.3.3260.1411
Fax: 81.3.3260.7100
Technical Center
Tel: 81.471.43.8200

Marubeni Solutions
1-26-20 Higashi
Shibuya-ku, Tokyo 150-0001
Tel: 81.3.5778.8662
Fax: 81.3.5778.8669

Shinki Electronics
Myuru Daikanyama 3F
3-7-3 Ebisu Minami
Shibuya-ku, Tokyo 150-0022
Tel: 81.3.3760.3110
Fax: 81.3.3760.3101

Yokohama-City
Innotech
2-15-10 Shin Yokohama
Kohoku-ku
Yokohama-City, 222-8580
Tel: 81.45.474.9037
Fax: 81.45.474.9065

Macnica Corporation
Hakusan High-Tech Park
1-22-2 Hadusan, Midori-Ku,
Yokohama-City, 226-8505
Tel: 81.45.939.6140
Fax: 81.45.939.6141

The Netherlands
Eindhoven
Acal Nederland b.v.
Beatrix de Rijkweg 8
5657 EG Eindhoven
Tel: 31.40.2.502602
Fax: 31.40.2.510255

Switzerland
Brugg
LSI Logic Sulzer AG
Mattenstrasse 6a
CH 2555 Brugg
Tel: 41.32.3743232
Fax: 41.32.3743233

Taiwan
Taipei
Avnet-Mercuries Corporation, Ltd
14F, No. 145,
Sec. 2, Chien Kuo N. Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2516.7303
Fax: 886.2.2505.7391

Lumax International Corporation, Ltd
7th Fl., 52, Sec. 3
Nan-Kang Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2788.3656
Fax: 886.2.2788.3568

Prospect Technology Corporation, Ltd
4Fl., No. 34, Chu Luen Street
Taipei, Taiwan, R.O.C.
Tel: 886.2.2721.9533
Fax: 886.2.2773.3756

Wintech Microeletronics Co., Ltd
7F., No. 34, Sec. 3, Pateh Road
Taipei, Taiwan, R.O.C.
Tel: 886.2.2579.5858
Fax: 886.2.2570.3123

United Kingdom
Maidenhead
Azzurri Technology Ltd
16 Grove Park Business Estate
Waltham Road
White Waltham
Maidenhead, Berkshire SL6 3LW
Tel: 44.1628.826826
Fax: 44.1628.829730

Milton Keynes
Ingram Micro (UK) Ltd
Garamonde Drive
Wymbush
Milton Keynes
Buckinghamshire MK8 8DF
Tel: 44.1908.260422

Swindon
EBV Elektronik
12 Interface Business Park
Bincknoll Lane
Wootton Bassett,
Swindon, Wiltshire SN4 8SY
Tel: 44.1793.849933
Fax: 44.1793.859555

♦ Sales Offices with Design Resource Centers