LSI LOGIC

L64360 and ATMizer™ Architecture Technical Manual

This document is preliminary. As such, it contains data derived from functional simulations and performance estimates. LSI Logic has not verified either the functional descriptions, or the electrical and mechanical specifications using production parts.

Document MN71-000101-99 A, First Edition (February 1995)
This document applies to Revision A of the L64360 and the ATMizer™ Architecture and to all subsequent versions unless otherwise indicated in a subsequent edition or an update to this edition of the document.

Publications are stocked at the address given below. Requests should be addressed to:

LSI Logic Corporation
Literature Distribution, M/S D-102
1551 McCarthy Boulevard
Milpitas, CA 95035
Fax: 408.433.8989

Preface

This book is the primary reference and technical manual for the L64360 chip and the ATMizer Architecture upon which it is based. It contains a complete functional description of the L64360 and the ATMizer Architecture and includes complete physical and electrical specifications for the L64360.

Audience

This book assumes that the reader has some familiarity with microprocessors and related support devices. This book is written for:

- Engineers and managers who are evaluating the L64360 or the ATMizer Architecture for possible use in a system
- Engineers who are designing the L64360 or the ATMizer Architecture into a system

Organization

This book has the following chapters and appendices:

- Chapter 1, **Introduction**, provides an overview of the ATMizer Architecture, describes some ATMizer Architecture applications, and lists the ATMizer Architecture's features.
- Chapter 2, **Functional Overview**, provides a functional overview of the ATMizer Architecture and the L64360 implementation.
- Chapter 3, **Signal Descriptions**, describes the signals that comprise the bit-level interface to the L64360.
- Chapter 4, **ATMizer Processing Unit (APU) and Prefetch Buffer**, describes the function and operation of the ATMizer Processing Unit and the Prefetch Buffer.
- Chapter 5, **Instruction RAM (IRAM) and Serial Interface**, describes the function and operation of the Serial Interface and how to load the Instruction RAM.
- Chapter 6, **Virtual Channel RAM (VCR)**, describes the function and operation of the Virtual Channel RAM.

- Chapter 7, **Pacing Rate Unit (PRU)**, describes the function and operation of the Pacing Rate Unit.
- Chapter 8, **DMA Controller (DMAC)**, describes the function and operation of the DMA Controller, which is contained within the Host/DMA Port.
- Chapter 9, **ATM Cell Interface (ACI)**, describes the function and operation of the ATM Cell Interface.
- Chapter 10, **Secondary Port (SP)**, describes the function and operation of the Secondary Port.
- Chapter 11, **System Mapping**, describes the ATMizer Architecture system hardware map.
- Chapter 12, **Operation**, describes the ATMizer Architecture operation.
- Chapter 13, **Functional Waveforms**, contains and describes the ATMizer Architecture functional waveforms.
- Chapter 14, **Registers**, describes and summarizes all the ATMizer Architecture registers.
- Chapter 15, **Specifications**, describes the electrical and mechanical characteristics of the L64360.
- Appendix A, **Glossary of Abbreviations**, provides a glossary of abbreviations that are used in this manual.
- Appendix B, **Customer Feedback**, provides a form that you may use to fax LSI Logic your comments on the content and quality of this manual.

Related Publications

CW33300 Enhanced Self-Embedding™ Processor Core User's Manual, Order No. C14014

LR33300 and LR33310 Self-Embedding™ Processors User's Manual, Order No. J14028

Conventions Used in this Manual

The first time a word or phrase is defined in this manual, it is *italicized.*

The following signal naming conventions are used throughout this manual:

- A level-significant signal that is true or valid when the signal is LOW always has an overbar ($\overline{}$) over its name.
- An edge-significant signal that initiates actions on a HIGH-to-LOW transition always has an overbar ($\overline{}$) over its name.

The word *assert* means to drive a signal true or active. The word *deassert* means to drive a signal false or inactive.

Hexadecimal numbers are indicated by the prefix “0x” before the number—for example, 0x32CF. Binary numbers are indicated by a subscripted “2” following the number—for example, $0011.0010.1100.1111_2$.

Contents

Tables

Chapter 1
Introduction

This chapter provides an overview and lists the features of the ATMizer Architecture and the L64360 chip.

This chapter has three sections:

- Section 1.1, “Overview”
- Section 1.2, “Features”
- Section 1.3, “Applications”

1.1 Overview

The ATMizer Architecture from LSI Logic provides a powerful, flexible solution to Asynchronous Transfer Mode (ATM) network control. The ATMizer Architecture is a group of carefully chosen hardware functional blocks that provide complete control over ATM Segmentation and Reassembly (SAR) and all ATM Layer operations. The architecture also includes the ATM Processing Unit (APU), a 32-bit RISC CPU (based on LSI Logic’s MIPS R3000 architecture), which, through user firmware, controls the functional blocks. This combination of hardware function and software control allows the user to dynamically solve any ATM Networking problem. The ATMizer Architecture enables market leaders to deploy unique ATM products today, while working with the standard as it evolves. The on-chip processing power allows users to simply download new code to accommodate changes to ATM standards or congestion control algorithms.

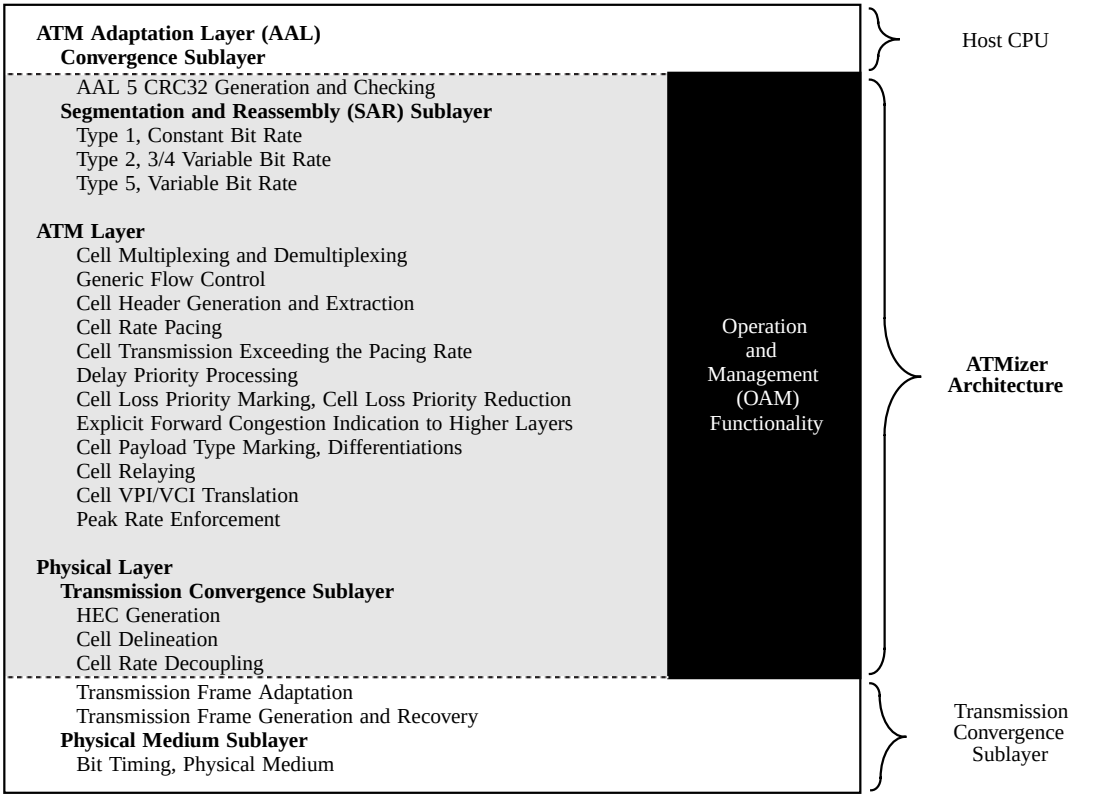
The ATMizer Architecture can either be embedded in Customer Specific Integrated Circuits (CSICs) or Application Specific Standard Products (ASSPs), or it can be used stand-alone as the L64360 chip.

The ATMizer Architecture can also perform management of Convergence Sublayer-Protocol Data Unit (CS-PDU) link lists (lists of CS-PDUs in need of segmentation), cell switching, scatter-gather DMA, memory buffers (in scatter-gather implementations), ATM header manipulation, traffic

shaping, congestion control, error monitoring, statistics gathering, host messaging, diagnostics, Virtual Channel Identifier (VCI) translation, and Virtual Path Identifier (VPI) translation. It also can transmit and receive various cell sizes, which enables users to differentiate system cell structures. In addition, it can perform Operation and Management (OAM) functions for all of the supported layers in firmware.

The ATMizer Architecture handles the Broadband-Integrated Service Digital Network (B-ISDN) layers highlighted in Figure 1.1.

Figure 1.1
ATMizer Architecture
Supported B-ISDN Layers

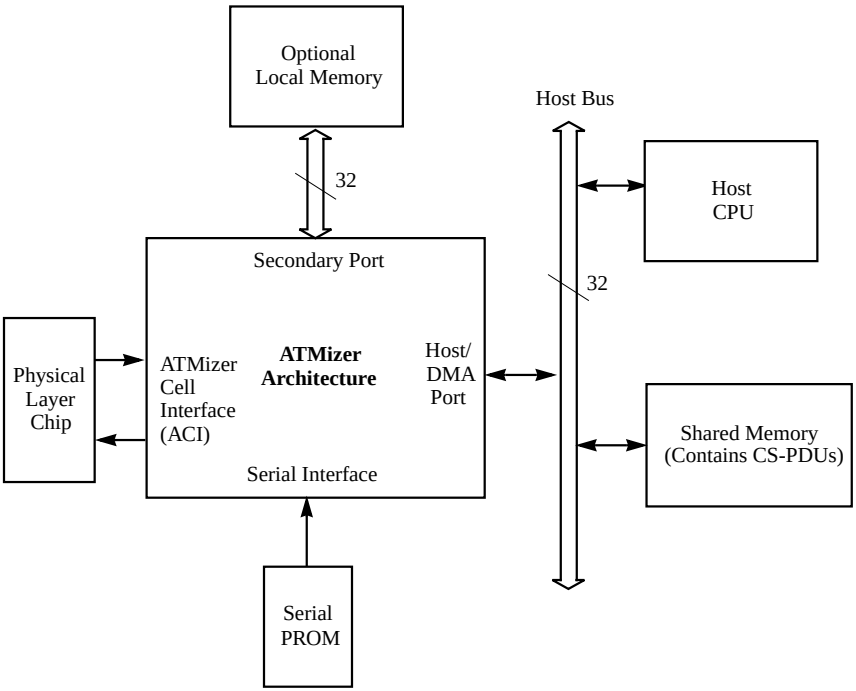


By supporting AALs 1, 2, 3/4, and 5, the ATMizer can process up to 64K Virtual Channels (VCs) of voice, data and video simultaneously. This combination of maximum VC support, programmability, and simultaneous

support of all data types provides unmatched flexibility for the emerging ATM standard.

Figure 1.2 shows a typical ATMizer Architecture with supporting logic.

Figure 1.2 ATMizer Architecture with Support Logic



The embedded APU enables customers to immediately realize ATM termination in a single device, and subsequently update signaling, congestion, and traffic management algorithms in firmware as the ATM standard evolves. By executing realtime instructions from an internal instruction RAM, the intelligent APU also enables ATM systems to respond immediately to network congestion without Host CPU intervention. The ATMizer Architecture couples the flexibility of a microprocessor with the economies of a single-chip solution, enabling users to differentiate their products in both hardware and software.

The ATMizer Architecture dramatically improves price/performance over alternative solutions by incorporating a wide range of system-level capabilities on a single chip, including:

- an eight-byte buffer on the transmitter and receiver to eliminate the need for buffers when interfacing to the Physical Layer

- a 4-Kbyte Virtual Channel RAM to provide additional cell buffering and to minimize external memory requirements
- a powerful 32-bit scatter-gather DMA engine that handles contiguous and noncontiguous CS-PDU protocols for better utilization of system memory
- a Secondary Port to access external memory-mapped devices and Channel Parameters when the DMA engine is busy via prefetch buffers

The ATMizer Architecture also provides a generic physical interface compatible with most physical interfaces, including the proposed UTOPIA Interface through an eight-bit parallel cell interface. The Host/DMA Port provides a 32-bit VLBus-like interface from the on-chip linked-list Direct Memory Access Controller (DMAC).

1.2 Features

The ATMizer Architecture features have been divided into five sections:

- General Features
- ATM Adaptation Layer Features
- ATM Layer Features
- ATM Cell Interface (ACI) Features
- Diagnostic Support Features

General Features

The ATMizer Architecture general features are:

- Supports ATM data rates of up to 155.52 Mbits/second
- Simultaneously supports ATM Adaptation Layers 1, 2, 3/4, and 5
- Handles contiguous and non-contiguous CS-PDUs
- 32-bit DMA addressing capabilities and 32-bit DMA data interfacing capabilities
- Implements Peak Rate Pacing, Maximum Burst Length, and Global Pacing for aggregate traffic shaping
- Supports up to 65536 VCs
- Supports simultaneous segmentation and reassembly of some VCs while cell switching others

- Implements a user-programmable 32-bit MIPS RISC CPU (ATMizer Processing Unit) controls all aspects of the ATM cell generation and switching processes

  The APU controls:

  - Scatter-gather DMA Algorithms
  - AAL SAR-PDU Header and Trailer Generation
  - ATM Header Generation and Manipulation
  - Host Interrupt/Messaging
  - Error Handling
  - Congestion Control
  - Statistics Gathering
  - Diagnostic Operation
  - User-defined Functions
- Internal caching of data structures, buffer link lists, and messages in 4-Kbyte Virtual Channel RAM (VCR), coupled with received cell buffers, allows for the development of memory-less network interface cards (all CS-PDUs undergoing segmentation and reassembly reside in system memory)
- General purpose 32-bit (address/data multiplexed) Secondary Port Interface with 4-Mbyte address space
- Four-word Prefetch Buffer
- Four-word Write Buffer
- Supports atomic (read-modify-write) transactions and Host/DMA Port CPU steal cycle
- Direct connection to the Universal Test & Operations Cell-based Physical layer (PHY) Interface for ATM (UTOPIA) and Standard ATM Interface (SAI)
- Supports cell size transformation for use in application that converts standard ATM cell into switch specific format

ATM Adaptation Layer Features

The ATM Adaptation Layer features are:

- AAL 1 cell generation from realtime data-streams including SAR Header generation and Residual Time Stamp Insertion
- Supports simultaneous segmentation and reassembly of AAL 2, 3/4, and 5 CS-PDUs, and AAL 1 cell generation from realtime datastreams
- Scatter-gather capabilities on reassembly and segmentation (implemented by user firmware)

  CS-PDUs need not be contiguous in system memory which allows for efficient use of memory space, higher throughput (no moves necessary to form contiguous CS-PDUs), and low latency attributable to devices such as routers
- Higher-layer header extraction and data alignment capabilities for application acceleration
- CRC10 generation and checking for AAL 3/4 SAR-PDUs
- CRC32 generation and checking for AAL 5 CS-PDUs segmentation and reassembly

ATM Layer Features

The ATM Layer features are:

- ATM Header generation and manipulation
- Support for VCI/VPI translation and cell switching
- Support for a user-defined cell size up to 65 bytes to allow for the prepending of a switch-specific header
- 10 Peak Rate Pacing Counters
- Advanced congestion control capabilities

  User firmware specified congestion control algorithms provide for immediate reaction to congestion notification. Fast response (within one cell time) results in fewer cells sent into a congested network, minimizing cell loss and CS-PDU retransmissions resulting in higher overall throughput. Congestion control routines are part of user firmware and can be modified as more is learned about congestion in actual ATM networks.
- Cell Loss Priority marking and manipulation (with AAL 5 High-Medium-Low Priority CS-PDU support)

ATM Cell Interface (ACI) Features

The ATM Cell Interface (ACI) features are:

- Frequency decoupling logic
- Eight-bit parallel transmit-data output bus
- Eight-bit parallel receive-data input bus
- Eight-byte buffers in transmitter and receiver that allows direct connection of data output and input buses to transceivers (no external buffering required)
- Global Pacing Rate Register that allows the APU to set the percentage of Idle Cells to be sent over the ACI, which enables aggregate traffic shaping and is a quick way of reducing data speeds upon congestion notification. The system can gradually return to full speed operation under APU control.
- All buffering and metastability issues are dealt with inside the ATMizer Architecture
- Automatic Cell Rate Decoupling through Idle Cell insertion
- Separate transmitter and receiver data transfer acknowledgment input signals provide for operation with free running transmitter and receiver clocks (allows connection to Transmission Convergence Sublayer framing logic that requires gaps in the assigned cell stream for the insertion/extraction of framing overhead)
- Internal received cell buffers (4, 8, 16 or 32 cells) add a second layer of buffering between the ACI and main memory

  This buffering allows the ATMizer Architecture to absorb periods of high latency to main memory or long exception handling routines without losing received cells. This is especially important in memoryless network add-in cards for PCs and Workstations where the computer's main memory is the ATMizer Architecture's working memory space.
- Conformance to the UTOPIA proposed standard (Version 1.22)
- Programmable Header Error Control (HEC) generation and checking

Diagnostic Support Features

The diagnostic support features are:

- User firmware controlled statistics gathering capabilities (keeps track of any statistics the application or network management architecture requires)
- CRC10 and CRC32 error statistics gathering
- CRC32 errors forced for diagnostic purposes
- Diagnostic firmware downloaded to APU to aid system level diagnostics when troubleshooting system or line failures

1.3 Applications

Figure 1.3 shows one possible application for the L64360, a network interface card with no local memory.

The sections following the figure discuss ATM network issues:

- Scatter-Gather DMA
- Application Acceleration
- Cell Switching
- Congestion Control
- AAL 1 Realtime Data Streams
- Diagnostic Operation

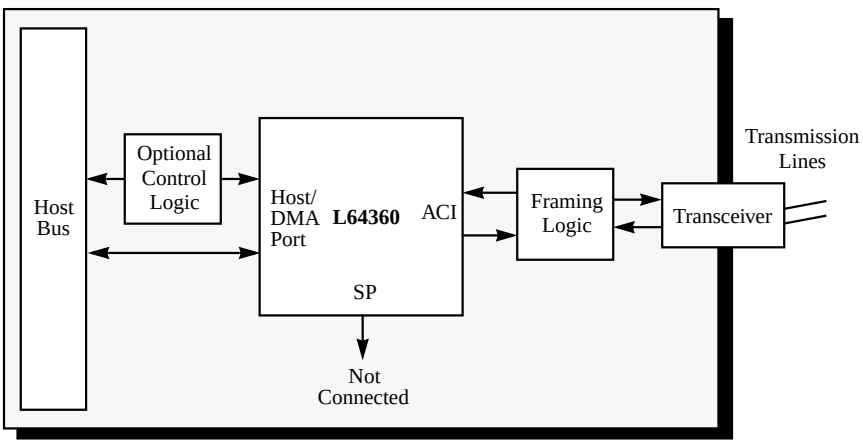


Figure 1.3 Network Interface Card with No Local Memory

Scatter-Gather DMA	The APU can execute segmentation and reassembly routines written by the system designer that perform scatter (segmentation) and gather (reassembly) of noncontiguous data structures (data structures that logically form a single CS-PDU) as is done in a Scatter-Gather DMA controller. The user can supply routines that handle ATM Adaptation Layer (AAL) and ATM header generation and extraction as well as link-list pointer management and buffer allocation.
Application Acceleration	Some applications can be accelerated by header stripping and data alignment. The ATMizer Architecture can aid network software by stripping higher-layer headers from incoming CS-PDUs and placing them in specific memory locations. In addition, the ATMizer Architecture can utilize the powerful byte-alignment capabilities of its DMA Controller to ensure that the user-data payload portion of the higher-layer PDU is written into memory word-aligned. This procedure releases application-layer software from the responsibility of ensuring proper data alignment.
Cell Switching	The ATMizer Architecture allows you to terminate all Virtual Channels (VCs) or terminate some VCs while switching others. On a per-VC basis the APU can make a determination as to whether it should reassemble the Segmentation and Reassembly (SAR) User-Payload into a CS-PDU or simply pass the entire cell, headers and trailers intact, to some other memory-mapped ATM port or ATM switch interface. The ATMizer Architecture can even switch cells between its receiver and transmitter without touching system memory. This implementation structure can be put to use in ring, dual, and triple-port switching fabrics, and other topologies. The APU can make cell-switching decisions (whether to translate the VCI, the VPI, or both, or neither, or whether to do multicast expansion) in real-time and then perform the operations. Furthermore, in switching applications, the ATMizer Architecture can support a user cell size of up to 65 bytes, which allows the user to include up to 12 bytes of switch-specific information to each cell.

Congestion Control

No one has seen enough ATM networks in operation to gain a real understanding of ATM network congestion. As the industry moves ahead with ATM, it is reassuring to note that the ATMizer Architecture, with its user programmable CPU positioned directly at the ATM line interface, is capable of executing or facilitating almost any congestion control algorithm imaginable. And because user firmware is downloaded at system reset, systems in the field can be updated with new congestion control algorithms as more is learned about congestion in real ATM networks.

The ATMizer Architecture also offers fast congestion response time. Cells arriving at the ACI with notification of network congestion can affect the transmission of the very next cell, either inhibiting it altogether, slowing down the rate of transmission of assigned cells, or forcing Cell Loss Priority (CLP) reductions. The user provides the algorithm. The ATMizer Architecture provides the hardware pacing logic, aggregate traffic shaping capability, and the processor to execute the algorithm.

AAL 1 Realtime Data Streams

The APU performs the realtime data stream buffer (DS1, voice, video, etc.) transfers (limits on ATM data rates may apply). The AAL 1 Segmentation and Reassembly (SAR) Header requires a Residual Time Stamp (RTS). The AAL 1 segmentation routine running on the APU can access RTS values from any memory-mapped device or location and carefully interleave the RTS value into the headers of the AAL 1 cell stream. When a new RTS value is needed, the APU retrieves it. When sequence numbers and sequence number protection are called for, the APU generates and inserts the appropriate information into the SAR Header. And on reassembly, the APU will verify sequence number integrity and sequentially and pass the RTS value to the appropriate device.

Diagnostic Operation

The ATMizer Architecture can actively participate in diagnostic operations by forcing CRC32 errors, gathering line statistics, and user defined operations. And under normal operating conditions, the APU can be chartered with the additional task of statistics gathering to aid network management process. All of these operations are made possible by the inclusion of a user-programmable APU.

Chapter 2
Functional Overview

This chapter provides a functional overview of the ATMizer Architecture and the L64360 chip.

This chapter has three sections:

- Section 2.1, “Overview”
- Section 2.2, “Functional Blocks”
- Section 2.3, “Buses”

2.1 Overview

The ATMizer Architecture provides ATM system designers with a segmentation and reassembly architecture that can, through user firmware control, implement ATM end stations and switching stations in a number of different ways. The ATMizer Architecture provides a number of critical hardware functions that are controlled by the firmware that a user downloads to the ATMizer Architecture APU at system reset time.

2.2 Functional Blocks

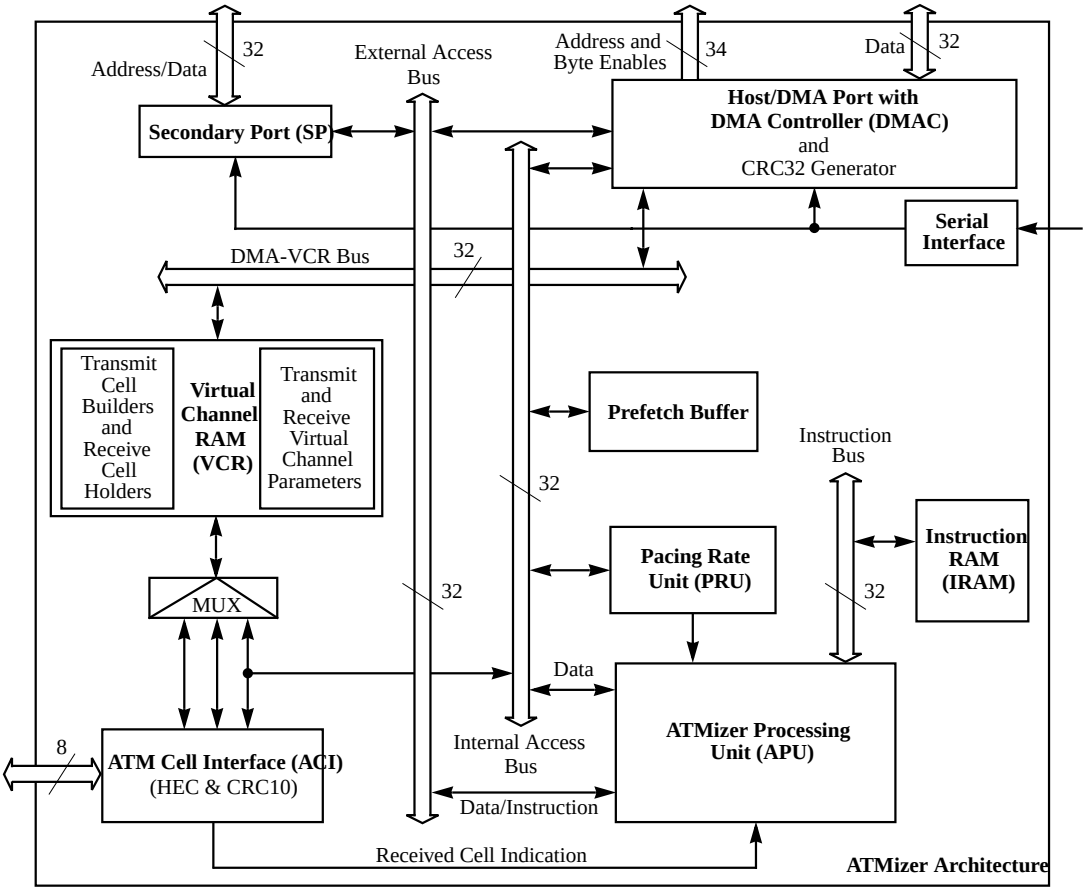
The ATMizer Architecture contains the following functional blocks:

- ATMizer Processing Unit (APU) and Prefetch Buffer
- Instruction RAM (IRAM) and Serial Interface
- Virtual Channel RAM (VCR)
- Pacing Rate Unit (PRU)
- DMA Controller (DMAC)
- ATM Cell Interface (ACI)
- Secondary Port (SP)

Figure 2.1, the ATMizer Architecture functional block diagram, shows the relationship of the buses and the hardware functions that firmware controls

within the ATMizer Architecture. Chapters 4 through 10 provide further description of these functional blocks.

Figure 2.1
ATMizer Architecture Functional Block Diagram



2.3 Buses

The ATMizer Architecture contains the following four buses that provide powerful solutions for many ATM applications:

- DMA-VCR Bus

The DMA-VCR Bus is a dedicated 32-bit data and 12-bit address connection from the DMA Controller to one of the read/write ports of the VCR.

- Internal Access Bus

  The Internal Access Bus connects the internal devices such as registers, VCR, and Prefetch Buffer to the APU. The Internal Access Bus shares the second VCR Port with the ACI. Since the Internal Access Bus is separate from the DMA-VCR Bus, the APU can access the internal registers and the VCR while the DMA is active.

- External Access Bus

  The External Access Bus enables the APU to perform direct load and store operations from external devices. A load or store from the Secondary Port can be done while the DMA is active. For example, the APU can program the DMA Controller during segmentation to bring in a 48-byte SAR-PDU and immediately after programming the DMA Register do a load or a store from the Secondary Port.

- Instruction Bus

  The Instruction Bus enables the APU to perform instruction fetches from the IRAM. The APU can also execute instructions from external devices if firmware exceeds 1 Kword.

Chapter 3
Signal Descriptions

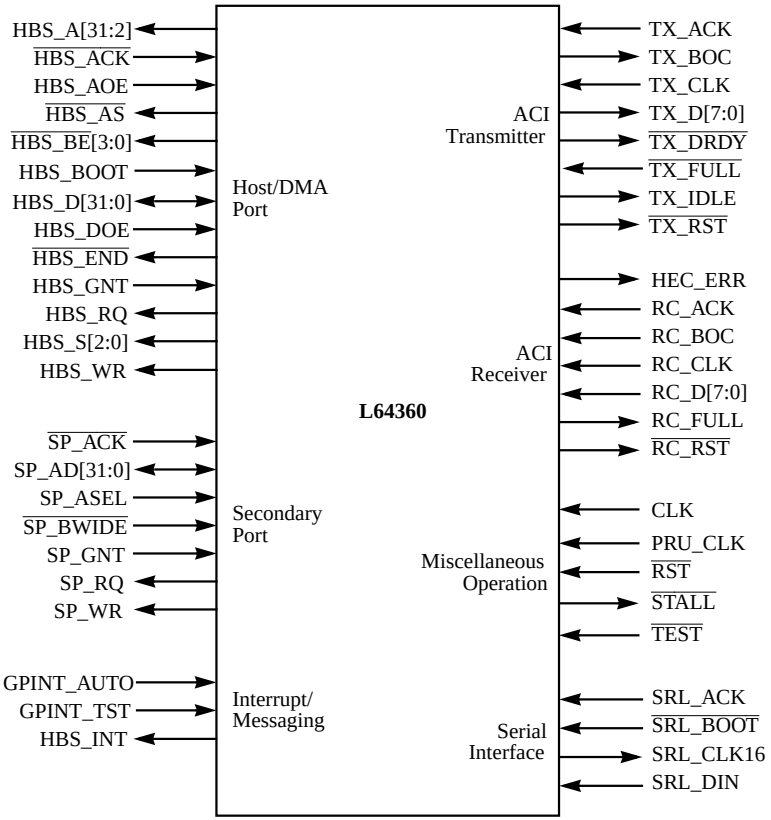
This chapter describes the signals that comprise the bit-level interface of the L64360 chip.

This chapter has eight sections:

3.1 L64360 Logic Symbol

Figure 3.1 shows the L64360 signals, their direction, and their polarity.

Figure 3.1
L64360 Logic Symbol



3.2 Host/DMA Port

These signals interface the L64360 DMA Controller and APU to the Host Bus. For more information on the DMAC see Chapter 8.

HBS_A[31:2] **Host/DMA Port Address Bus** **Output**
During DMA or CPU operations, the Host /DMA Port drives this address bus to provide L64360 addresses to system components. Each time $\overline{\text{HBS\_ACK}}$ is asserted, the word address HBS_A[23:2] is incremented. HBS_A[31:24] are never incremented, so user firmware should never initiate burst operations that cross 16-Mbyte boundaries. After a system reset, the L64360 3-states HBS_A[31:2].

$\overline{\text{HBS\_ACK}}$ **Host/DMA Port Data Acknowledgment** **Input**
Asserting $\overline{\text{HBS\_ACK}}$ LOW indicates that there is valid data on HBS_D[31:0]. During Host/DMA Port writes to an external device, the device should assert $\overline{\text{HBS\_ACK}}$ LOW to inform the

L64360 that it is going to latch the data for the current write operation at the next rising edge of CLK. If the acknowledged transfer was the last transfer of the operation, the L64360 deasserts HBS_RQ following the rising edge of CLK.

HBS_AOE **Host/DMA Port Address Output Enable** **Input**

Asserting HBS_AOE HIGH, enables the Host/DMA Port Address Bus (HBS_A[31:2]) and the control signals ($\overline{\text{HBS\_AS}}$, $\overline{\text{HBS\_BE}}$[3:0], $\overline{\text{HBS\_END}}$, HBS_WR, and HBS_S[2:0]). Deasserting HBS_AOE LOW, 3-states the address bus and control signals.

$\overline{\text{HBS\_AS}}$ **Host/DMA Port Address Strobe** **Output**

At the beginning of each transaction, after the L64360 has granted the bus (HBS_GNT is asserted), L64360 asserts $\overline{\text{HBS\_AS}}$ for one clock cycle to indicate that HBS_A[31:2] and $\overline{\text{HBS\_BE}}$[3:0] are valid. After a system reset, the L64360 3-states $\overline{\text{HBS\_AS}}$.

$\overline{\text{HBS\_BE}}$[3:0] **Host/DMA Port Byte Enables** **Output**

The L64360 enables each of the four bytes of HBS_D[31:0] by asserting the corresponding bit of $\overline{\text{HBS\_BE}}$[3:0] LOW. When accessing a single byte, only one of the $\overline{\text{HBS\_BE}}$[3:0] signals is asserted to indicate valid data on the corresponding byte port. During a 32-bit word access from an external device, the L64360 Host/DMA Port asserts all of the $\overline{\text{HBS\_BE}}$[3:0] signals LOW and drives the appropriate word address onto HBS_A[31:2]. The table below maps the $\overline{\text{HBS\_BE}}$[3:0] signals to their corresponding bytes on HBS_D[31:0].

$\overline{\text{HBS\_BE}}$[3:0] Bit	***HBS_D[31:0] Byte***
$\overline{\text{HBS\_BE3}}$	HBS_D[31:24]
$\overline{\text{HBS\_BE2}}$	HBS_D[23:16]
$\overline{\text{HBS\_BE1}}$	HBS_D[15:8]
$\overline{\text{HBS\_BE0}}$	HBS_D[7:0]

After a system reset, the L64360 3-states $\overline{\text{HBS\_BE}}$[3:0].

HBS_BOOT **Host/DMA Port Boot Select** **Input**

HBS_BOOT selects the port used for booting. Asserting HBS_BOOT HIGH selects the Host/DMA Port. Deasserting HBS_BOOT LOW selects the Secondary Port.

HBS_D[31:0] **Host/DMA Port Data Bus** **Bidirectional**

During read operations, the Host/DMA Port samples HBS_D[31:0] on the rising edge of CLK when $\overline{\text{HBS\_ACK}}$ is asserted.

During write operations, the Host/DMA Port sources data onto HBS_D[31:0]. The Host/DMA Port responds to $\overline{\text{HBS\_ACK}}$ by sourcing data for the next transfer onto HBS_D[31:0].

HBS_DOE **Host/DMA Port Data Output Enable** **Input**

Asserting HBS_DOE HIGH, enables output on the Host/DMA Port Data Bus, HBS_D[31:0]. Deasserting HBS_DOE LOW 3-states the data bus.

$\overline{\text{HBS\_END}}$ **Host/DMA Port Operation Ending** **Output**

During a burst operation, the Host/DMA Port asserts $\overline{\text{HBS\_END}}$ LOW, when it detects the second-to-the-last $\overline{\text{HBS\_ACK}}$, to warn the memory controller that the current transfer will end the next time $\overline{\text{HBS\_ACK}}$ is asserted LOW. $\overline{\text{HBS\_END}}$ is deasserted following the rising clock edge during which the L64360 samples the final transfer acknowledgment ($\overline{\text{HBS\_ACK}}$ asserted) for the given burst operation. $\overline{\text{HBS\_END}}$ is also asserted when the DMA is suspended by an APU transaction. After a system reset, the L64360 3-states $\overline{\text{HBS\_END}}$.

HBS_GNT **Host/DMA Port Grant Operation** **Input**

In response to HBS_RQ asserted, the bus arbiter asserts HBS_GNT HIGH to notify the L64360 that it has been granted the bus so a DMA or APU operation may start. If no other Host CPU requests the bus, the External Bus Arbiter can grant the bus to the L64360 all the time, eliminating one access cycle.

HBS_RQ **Host/DMA Port Operation Request** **Output**

The L64360 asserts HBS_RQ HIGH to request access to the device attached to the Host/DMA Port. The external bus arbiter should respond to HBS_RQ by asserting HBS_GNT, which allows the L64360 to proceed with the transfer.

If the APU has queued up back-to-back DMA Operations (it may have even entered a write busy stall because it attempted to write a new initialization word to a busy DMAC), the L64360 does not deassert HBS_RQ in response to the final Host/DMA Port Acknowledge and the next operation begins immediately. The external device should check the state of $\overline{\text{HBS\_END}}$ to

distinguish between DMA operation boundaries. After a system reset, the L64360 drives HBS_RQ LOW.

HBS_S[2:0] **Host/DMA Port DMA Transfer Size** **Output**

These three signals are the encoded value of the DMA transfer size. They are used when the user attaches the L64360 to an SBus. The following table shows combinational values for HBS_S[2:0].

HBS_S2	*HBS_S1*	*HBS_S0*	*DMA Transfer Size (Bytes)*
0	0	0	4
0	0	1	Not Supported
0	1	0	Not Supported
0	1	1	Not Supported
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	8

After a system reset, the L64360 3-states HBS_S[2:0].

HBS_WR **Host/DMA Port Operation Type** **Output**

HBS_WR qualifies the type of operation requested by the L64360 asserting HBS_RQ. The L64360 asserts HBS_WR HIGH when the L64360 is initiating a write. The L64360 deasserts HBS_WR LOW when the L64360 is initiating a read. After a system reset, the L64360 3-states HBS_WR.

3.3 ACI Transmitter

These signals transmit ATM cell data and control ATM cell data transmission. For more information see Chapter 9.

TX_ACK **ACI Transmitter Data Acknowledgment** **Input**

The Transmission Convergence Sublayer (TCS) framing logic (external to the L64360) asserts TX_ACK HIGH when it has sampled the data value on TX_D[7:0]. The L64360 responds to TX_ACK by placing the next byte onto TX_D[7:0]. If the next byte is the first byte of a new cell, the L64360 also asserts TX_BOC. The L64360 ACI Transmitter can be gapped by deasserting TX_ACK when TX_CLK is free-running, or by shutting TX_CLK off and on while constantly asserting TX_ACK HIGH. If the external logic is unable to sample the byte on TX_D[7:0] in a given cycle, it should deassert TX_ACK.

TX_BOC **ACI Transmitter Beginning of Cell** **Output**

The L64360 ACI Transmitter asserts TX_BOC HIGH while the first byte of a cell is sourced on TX_D[7:0]. TX_BOC is deasserted after the first TX_ACK is received. After a system reset, the L64360 drives TX_BOC LOW.

TX_CLK **ACI Transmitter Clock** **Input**

All transmission signals are sourced or sampled on the rising edge of this clock. TX_CLK drives the buffer inside the Transmitter portion of the L64360 ATM Cell Interface. All data transfers from the L64360 over TX_D[7:0] are synchronized to this clock, as are $\overline{\text{TX\_DRDY}}$, TX_BOC, and TX_IDLE. Logic inside the L64360 synchronizes the L64360 System Clock and the ACI Transmitter data buffer circuitry, which is sequenced off of TX_CLK. The system designer need not worry about metastability at the transmitter output. TX_CLK is the byte clock of the external transmitter and can be operated at any frequency less than or equal to half the System Clock.

TX_D[7:0] **ACI Transmitter Data Bus** **Output**

The L64360 sources byte-aligned cell data onto TX_D[7:0]. Bit 7 is the first bit to be transmitted over the serial line. After a system reset, TX_D[7:0] are undefined.

$\overline{\text{TX\_DRDY}}$ **ACI Transmitter Data Ready** **Output**

The L64360 asserts $\overline{\text{TX\_DRDY}}$ LOW three cycles after it deasserts $\overline{\text{TX\_RST}}$ HIGH. The L64360 asserts $\overline{\text{TX\_DRDY}}$ and TX_BOC to indicate that external logic can sample TX_D[7:0] and issue an acknowledge (TX_ACK). Once asserted, $\overline{\text{TX\_DRDY}}$ remains asserted until the next system or transmitter reset or until $\overline{\text{TX\_FULL}}$ is asserted. After a system reset, the L64360 3-states $\overline{\text{TX\_DRDY}}$.

$\overline{\text{TX\_FULL}}$ **TCS Receive Buffer Full** **Input**

This signal is provided for the UTOPIA Interface. When the TCS Receiver Buffer is full, it notifies the ACI by asserting $\overline{\text{TX\_FULL}}$. The ACI responds by deasserting $\overline{\text{TX\_DRDY}}$, which should be connected directly to the Physical Layer enable signal.

TX_IDLE **Transmitting Idle Cell** **Output**

The L64360 ACI Transmitter asserts TX_IDLE HIGH when the next outgoing cell is an Idle Cell. Transmission convergence framing logic that does not handle Idle Cells must still assert TX_ACK until the entire Idle Cell passes. After a system reset, the L64360 3-states TX_IDLE.

$\overline{\text{TX\_RST}}$ **Transmitter Reset** **Output**

Asserting $\overline{\text{TX\_RST}}$ resets the Physical Layer. After the L64360 is powered on, the L64360 asserts $\overline{\text{TX\_RST}}$ LOW within two to four cycles of the L64360 System Clock. $\overline{\text{TX\_RST}}$ is deasserted two clock cycles after the Transmit Initialize Bit in the System Control Register is set to one. After a system reset, the L64360 drives $\overline{\text{TX\_RST}}$ LOW.

3.4 ACI Receiver

These signals receive ATM cell data and control ATM cell data reception. For more information see Chapter 9.

HEC_ERR **HEC Error** **Output**

The L64360 asserts HEC_ERR HIGH when the HEC Field that is received (Byte 5 of a cell) does not equal the HEC Field that the L64360 calculated from the ATM Header. HEC_ERR is only active when the receiver is configured to accept and check the HEC byte. After a system reset, the L64360 drives HEC_ERR LOW.

RC_ACK **ACI Receiver Data Acknowledgment** **Input**

Asserting RC_ACK HIGH indicates that valid data has been placed on RC_D[7:0]. Framing logic in the transmission convergence unit should assert RC_ACK HIGH when it has placed data on RC_D[7:0]. The L64360 responds to RC_ACK by sampling RC_D[7:0] on the rising edge of RC_CLK. The ACI Receiver can be gapped by deasserting RC_ACK if external logic is unable to supply a byte on RC_D[7:0] in a given cycle.

RC_BOC **ACI Receiver Beginning of Cell** **Input**

Asserting RC_BOC HIGH signals the beginning of a cell to the ACI Receiver. When the Physical Layer asserts RC_BOC, the ACI Receiver starts a counter to count the number of bytes in the incoming cell.

RC_CLK **ACI Receiver Clock** **Input**

All receive signals are sourced or sampled on the rising edge of this clock. RC_CLK drives the buffer inside the ATM Cell Interface Receiver. All data transfers over RC_D[7:0] to the L64360, as well as the assertion of all output signals, are synchronized to RC_CLK. Logic inside of the L64360 handles synchronization between the L64360 System Clock and the ACI Receive Data Buffer circuitry powered by RC_CLK. The system designer need not worry about metastability at the Receiver input. RC_CLK is likely to be the clock derived from the line data and

can be operated at any frequency less than or equal to half the System Clock.

RC_D[7:0] **ACI Receiver Data Bus** **Input**
The L64360 receives byte aligned cell data on RC_D[7:0]. The Transmission Convergence Sublayer (TCS) framing logic uses RC_D[7:0] to send byte aligned cell data to the L64360. Bit 7 is the first bit received over the serial line.

RC_FULL **ACI Receiver Cell Holder Buffer Full** **Output**
The L64360 asserts RC_FULL HIGH when the internal Received Cell Buffer (Holder) in the VCR is almost full (six bytes before it is full).

When the BM Bit in the System Control Register is zero (UTOPIA Mode), the L64360 receives only one more byte after asserting RC_FULL. When BM Bit in the System Control Register is one (SAI mode), the L64360 finishes receiving the cell completely after asserting RC_FULL. After a system reset, the L64360 drives RC_FULL LOW.

$\overline{\text{RC\_RST}}$ **ACI Receiver Reset** **Output**
Because several parameters have to be configured before the ACI can receive any cell, firmware controls the deassertion of $\overline{\text{RC\_RST}}$. Setting the RI Bit in the System Control Register to one deasserts $\overline{\text{RC\_RST}}$. After a system reset, the L64360 drives $\overline{\text{RC\_RST}}$ LOW.

3.5 Secondary Port

These signals control the Secondary Port, a 32-bit, multiplexed address and data bus port. For more information see Chapter 10.

$\overline{\text{SP\_ACK}}$ **Secondary Port Data Acknowledgment** **Input**
Asserting $\overline{\text{SP\_ACK}}$ LOW indicates that valid data will be read or written on SP_AD[31:0] in the next clock cycle. During a Secondary Port read operation, an external device should assert $\overline{\text{SP\_ACK}}$ LOW one cycle before valid data is available on SP_AD[31:0]. During a Secondary Port write operation, an external device should also assert $\overline{\text{SP\_ACK}}$ one cycle before it latches the write data on SP_AD[31:0].

SP_AD[31:0] **Secondary Port Address/Data Bus** **Bidirectional**
SP_AD[31:0] is a multiplexed address and data bus on the Secondary Port. When SP_ASEL is asserted HIGH, SP_AD[31:0]

contains the information shown in the table below. When SP_ASEL is deasserted LOW, SP_AD[31:0] contains data.

SP_AD Bits	*Use*	*Settings*
[31:28]	Byte Enables	Deasserted LOW to select for Write, All Asserted HIGH for Read
27	Not Used	Deasserted LOW
26	Access Type	Deasserted LOW for Data, Asserted HIGH for Instruction
25	Block Fetch Bit	Asserted HIGH
24	Atomic Bit	Asserted HIGH
[23:22]	Not Used	Deasserted LOW
[21:0]	Address	–

Deasserting SP_GNT, 3-states SP_AD[31:0]. The Byte Enables on the Secondary Port, SP_AD[31:28], only apply to write transactions. During read transactions, SP_AD[31:28] must be asserted HIGH.

The table below maps the address phase SP_AD[31:28] signals to their corresponding bytes on the data phase of SP_AD[31:0].

SP_AD[31:28] Bit (Address Phase)	*SP_D[31:0] Byte (Data Phase)*
SP_AD31	SP_AD[31:24]
SP_AD30	SP_AD[23:16]
SP_AD29	SP_AD[15:8]
SP_AD28	SP_AD[7:0]

When the Block Fetch Bit is set, the external logic must respond with the correct number of acknowledge cycles. For a description on the Atomic Operation, refer to Section 4.8, “Atomic Transactions.”

SP_ASEL **Secondary Port Address/Data Select** **Input**

Asserting SP_ASEL HIGH causes the L64360 to drive an address on SP_AD[31:0]. During a read operation, deasserting SP_ASEL causes the L64360 to 3-state SP_AD[31:0] so that external logic can drive the data onto SP_AD[31:0]. During a write operation, deasserting SP_ASEL causes the L64360 to drive SP_AD[31:0] with data.

$\overline{\text{SP\_BWIDE}}$ **Secondary Port Byte-wide Device** **Input**

Asserting $\overline{\text{SP\_BWIDE}}$ LOW, during read operations, indicates that the external device attached to the port is eight bits wide.

With $\overline{\text{SP\_BWIDE}}$ asserted, the Secondary Port executes four cycles with sequential byte addresses beginning with the effective address. The external device has to assert $\overline{\text{SP\_ACK}}$ along with $\overline{\text{SP\_BWIDE}}$ for all four byte accesses to guarantee the L64360's proper operation. The external device should provide data on SP_AD[7:0] when it asserts $\overline{\text{SP\_BWIDE}}$.

SP_GNT **Secondary Port Bus Grant** **Input**

Asserting SP_GNT HIGH, causes the L64360 to drive the address on SP_AD[31:0]. At the end of a transmission the L64360 stops driving SP_AD regardless of the state of SP_GNT. Deasserting SP_GNT LOW, 3-states SP_AD[31:0]. If SP_GNT is asserted continuously, external logic can then sample the correct address in the same cycle that SP_RQ is asserted.

SP_RQ **Secondary Port Access Request** **Output**

When it has sourced a valid address on SP_AD[31:0], the L64360 asserts SP_RQ HIGH to initiate an access to the Secondary Port. After a system reset, SP_RQ is undefined for two clock cycles, then the L64360 drives SP_RQ LOW.

SP_WR **Secondary Port Operation Type** **Output**

SP_WR is valid only when SP_RQ is asserted HIGH. The L64360 asserts SP_WR HIGH to indicate that it is requesting a Secondary Port write operation. The L64360 deasserts SP_WR LOW to indicate that it is requesting a Secondary Port read operation. After a system reset, the L64360 drives SP_WR LOW.

3.6 Interrupt/ Messaging

These signals control host messaging. The APU is based on the CW33300 described in *CW33300 Enhanced Self-Embedding Processor Core User's Manual.*

GPINT_AUTO **General Purpose APU Interrupt** **Input**

GPINT_AUTO is connected to APU Interrupt2. APU software can disable or enable interrupts as necessary.

GPINT_TST **L64360 Interrupt** **Input**

GPINT_TST is connected to the CpCond0 signal in the APU. External logic can assert GPINT_TST to alert the APU. Asserting GPINT_TST HIGH sets the Branch on CpCond0 instruction to TRUE.

GPINT_TST can be used for message passing. APU firmware samples this signal to determine if the Host has a message for the APU.

HBS_INT **Host Interrupt** **Output**

The L64360 asserts the Host Interrupt when it wishes to interrupt the Host. It is likely to be used as part of the messaging system. The interpretation of this signal is defined by user firmware. The L64360 may assert HBS_INT HIGH to indicate error conditions, congestion problems, CS-PDUs reassembled, or other conditions. The L64360 APU asserts HBS_INT by performing a store operation to the Host Interrupt Register (refer to Section 4.7, "ATMizer Architecture-to-Host Messaging"). HBS_INT remains valid for four clock cycles and is then deasserted. After a system reset, the L64360 drives HBS_INT LOW.

3.7 Serial Interface

The Serial Interface signals control serial downloads. For more information see Chapter 5.

SRL_ACK **Serial Acknowledge** **Input**

SRL_ACK controls the bit rate that is applied to the L64360 during serial downloads. When downloading from a serial PROM, asserting SRL_ACK HIGH causes the L64360 to latch the bit on SRL_DIN on the rising edge of SRL_CLK16.

$\overline{\text{SRL\_BOOT}}$ **Serial Boot Select** **Input**

Asserting $\overline{\text{SRL\_BOOT}}$ LOW enables the Serial Interface. Deasserting $\overline{\text{SRL\_BOOT}}$ HIGH disables the Serial Interface.

SRL_CLK16 **Serial Clock** **Output**

SRL_CLK16 is the clock rate for the serial download mode, the System Clock rate divided by 16. This signal clocks bits from a serial device used in serial downloads to the L64360. When using a serial PROM, the SRL_ACK must be asserted HIGH and SRL_CLK16 should be used to clock the bits into the L64360.

SRL_DIN **Serial Data Input** **Input**

SRL_DIN is the data input for the serial downloading mode. The first bit is Bit 31 Word 0, followed by Bit 30 Word 0, and so on.

3.8 Miscellaneous Operation

The Miscellaneous Operation signals drive the clocks and system reset.

CLK — **System Clock** — **Input**

The CLK input runs the APU, Host/DMA Port, the Secondary Port, the VCR, and much of the logic in the ACI. CLK does not affect the transfer of byte data to or from the L64360 over the ACI (ACI transactions are controlled by TX_CLK and RC_CLK). Supported frequencies on CLK are 25, 33, 40, and 50 MHz.

PRU_CLK — **Pacing Rate Unit Clock** — **Input**

The clock connected to the PRU_CLK pin must run at half or less the System Clock Frequency (CLK). The down counters associated with the ten PRPCs count down by one every clock tick. The Clock Select Bit in the PRU Configuration Register selects the clock inputs to the PRPCs to be either CLK or PRU_CLK. In most applications, the PRPC clock should run at the physical line frequency. In this case either the RC_CLK or TX_CLK can be connected directly to PRU_CLK.

$\overline{\text{RST}}$ — **System Reset** — **Input**

Asserting $\overline{\text{RST}}$ LOW initiates the master reset of the L64360. This signal also resets the ACI Transmitters and Receivers. This signal is asynchronous.

$\overline{\text{STALL}}$ — **APU Pipeline Stall** — **Output**

The L64360 asserts this signal LOW when the APU pipeline is stalled. The L64360 deasserts this signal HIGH when the APU is executing instructions.

$\overline{\text{TEST}}$ — **Test Mode** — **Input**

Asserting this signal LOW causes the L64360 to 3-state all output and bidirectional pins. In normal operation this signal should be tied to VDD.

Chapter 4
ATMizer Processing Unit (APU) and Prefetch Buffer

This chapter describes the function and operation of the ATMizer Processing Unit and the Prefetch Buffer.

This chapter has twelve sections:

4.1 APU Overview

The ATMizer Processing Unit (APU) is a 32-bit RISC CPU core based on the MIPS R3000 architecture. The APU includes a CPU, a four-word write buffer, and a cache controller. The powerful, user-programmable CPU gives the ATMizer Architecture many unique capabilities.

The *CW33300 Enhanced Self-Embedding Processor Core User’s Manual* describes the core which is the basis for the APU.

The APU processes every incoming cell and generates every outgoing cell. The APU provides the operational control necessary to support functions

such as SAR and cell switching of multiple AAL-type cells, scatter-gather memory management operations, intelligent congestion control algorithms, traffic statistics gathering, and robust ATMizer Architecture-to-Host messaging. The APU firmware builds cells, controls messages between the ATMizer Architecture and the Host CPU, and services channel sequencing. Cell building consists of SAR Header and Trailer generation, ATM Header retrieval from the Channel Parameter Entry (CPE) for the Virtual Channel (VC), ATM Header manipulation and insertion, and programming the DMA operation for SAR-PDU retrieval.

The user-written firmware controls most ATMizer Architecture functions, including the following:

- Header and Trailer Generation and Retrieval
- DMA
- Pacing Rate Unit (PRU) Configuration
- ACI Cell Queuing and Cell Processing
- Memory Allocation
- ATMizer Architecture-to-Host Messaging
- Atomic Transactions
- Host/DMA Port Priority
- Congestion Control
- APU External Access

Sections 4.2 through 4.11 describe how the APU controls these functions.

Note The CpCond and Interrupt signals are internal to the APU. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

4.2 Header and Trailer Generation and Retrieval

The APU firmware generates SAR Headers (AAL 1, 2, and 3/4) and Trailers (AAL 2 and 3/4) during segmentation. Firmware can program the ACI to generate and insert the CRC10 Field. SAR Header generation includes sequence number generation and checking as well as message type insertion and extraction (BOM, COM, EOM, and SSM). The APU also initiates the appropriate DMA operations to retrieve SAR-Service Data Unit (SDU) from memory-based realtime data buffers (AAL 1) or CS-PDUs. The APU also retrieves and manipulates ATM Headers, which includes modifying

PTI and CLP fields. For cells that are to be switched, the APU makes the initial switching decision based on information contained in the CPE for the VC, as well as VCI/VPI translation if specified in the CPE.

4.3 DMA

The DMA Controller (DMAC) within the ATMizer Architecture handles memory transactions between the VCR and Host memory. It manages the functions of main and local (VCR) memory address incrementing, byte count reduction, and byte alignments.

The APU initiates DMA operations to:

- retrieve SAR-SDUs during segmentation
- restore SAR-SDUs to their respective CS-PDUs during reassembly
- switch entire cells, headers and trailers intact, to other memory-mapped ATM ports during switching operations
- transfer SAR-SDUs to and from realtime data stream buffers in applications supporting AAL 1 circuit interfaces (such as T1 lines)

The APU sets the following parameters in the DMAC to initiate a DMA operation:

- Main Memory Starting Address and Byte Offset
- Local VCR Starting Address and Local Byte Offset
- Number of Bytes to be Transferred (less than or equal to 64)
- Transfer Direction (Read or Write)
- Ghost Bit to generate CRC32 for non-contiguous PDUs

Writing to the DMAC causes the DMAC to initiate the transfer. The APU user firmware should check to see if the DMAC is busy before issuing a DMA command to the ATMizer Architecture. If it is busy, it should wait. The APU can perform direct load/store operations from/to Host memory while the DMAC is busy (APU steal cycle).

4.4 Pacing Rate Unit (PRU) Configuration

The APU writes to the Pacing Rate Unit to implement Peak Rate Pacing Maximum Burst Length for traffic shaping during the segmentation process. The PRU consists of 10 Peak Rate Pacing Counters (PRPCs) and 10 Peak Rate Pacing Initialization Registers. The APU sets the initial count values in one of the 10 Peak Rate Pacing Initialization Registers and starts

the count operation. When one or more counters time out, the PRU informs the APU by asserting Interrupt1 or CpCond2.

When the Host has a CS-PDU ready for segmentation, it passes a data structure, called the Channel Parameter Entry (CPE), to the APU. The CPE describes how to segment the CS-PDU and can be kept either in the VCR or in external memory. In either case, the APU has a pointer to the CPE, selects one of the 10 PRPCs, and attaches the CPE pointer or the CPE itself to the PRPC. When a PRPC has elapsed, one or more CS-PDUs are ready for segmentation.

If one or more counter expires at the same time, the APU can read the Channel Group Credit Register (CGCR) to determine which PRPCs have expired.

4.5 ACI Cell Queuing and Cell Processing

The APU queues cells for transmission by writing the VCR Start Address of a cell into the Transmit Cell Address FIFO in the ACI Transmitter. If no cell address is present in the FIFO when an end of cell boundary is reached, the Transmitter automatically sends an Idle Cell. For received cells, the APU decides between cell switching and circuit termination for each VC.

The APU performs internal cell switching (cell switching between its receiver and transmitter) by passing the VCR addresses of a received cell targeted for internal switching to the Cell Address FIFO in the ACI (ATM Cell Interface) Transmitter.

The APU also sets the Global Pacing Rate Register (GPRR) in order to shape the assigned cell content of the outgoing cell stream. The GPRR is a simple count-down counter. When it reaches zero, The ACI Transmitter forces an Idle Cell to the external framing logic. The APU may also program the ACI to generate and insert the CRC10 Field for AAL 3/4 ATM cells and generates or checks the HEC Field.

4.6 Memory Allocation

During the reassembly process the APU manages the memory buffer. If memory is to be allocated to incoming CS-PDUs in fragments, the APU:

- tracks fragment boundaries
- issues additional fragments to CS-PDUs as needed
- generates link lists of the fragments allocated to a given CS-PDU

- sends messages from the ATMizer Architecture to the Host to inform the Host of CS-PDU completion, errors, and congestion problems

If the CS-PDU is not contiguous in the transmit direction, the APU firmware may have to recognize the difference between end-of-fragment boundaries and end-of-CS-PDU boundaries.

4.7 ATMizer Architecture-to-Host Messaging

The ATMizer Architecture does not enforce a particular messaging system between the APU and the Host system. Using the L64360, the user implements his own messaging system by polling the L64360 Interrupt input signal, GPINT_TST (connected directly to CpCond0 and tested with the Branch on CpCond0 True instruction), for an indication that the Host wishes to pass messages to the L64360.

The APU can read or write to or from any Host/DMA Port or Secondary Port memory-mapped location as part of a messaging mailbox system. The APU Interrupt input signal, GPINT_AUTO, can also be used in addition to or in place of GPINT_TST as part of the messaging system. Please note that GPINT_AUTO is a true interrupt and GPINT_TST is a polled condition input.

The L64360 asserts the Host/DMA Port Interrupt output signal, HBS_INT, to indicate to the Host that the L64360 wishes to or has already passed a message to the Host system. The L64360 asserts HBS_INT when the APU performs a store to the Host Interrupt Register. (The Host Interrupt Register is not really a register. It is an address decode circuit with an Effective Address of 0xFFF04B00.) Writing to the Host Interrupt Register causes the L64360 to assert HBS_INT for four clock cycles and then deasserts it. If the external system needs more than four cycles to recognize the interrupt, it must latch HBS_INT and then clear the latch when finished.

4.8 Atomic Transactions

The Secondary Port and the Host/DMA Port support atomic transactions (locked back-to-back read-modify-writes) in hardware. The APU can perform atomic transactions, although the MIPS architecture does not have a read-modify-write instruction. For the APU to do a read-modify-write, Bit 24 of the Effective Address must be set to one. In the Secondary Port, this bit is reflected on Bit 24 of the Physical Address during the address cycle. If the bit is set, it is the responsibility of the external arbiter to guarantee that the bus is not given to other master after the current operation is complete, because the L64360 performs the next operation in a locked

back-to-back manner. On the Host/DMA Port, if Bit 24 of the Effective Address is set, the L64360 asserts the Host/DMA Port Operation Request output signal, HBS_RQ until both transactions have finished.

When the APU firmware is initiating an atomic transaction on the Host/DMA Port, if the APU does not perform the second transaction (write) after it finishes the first transaction (read), the Host/DMA Port times-out 64 cycles after the acknowledge of the first transaction. The APU then gives up the bus by deasserting HBS_RQ, sets the Timeout Error Bit in the System Control Register, and asserts the APU internal Interrupt0 signal.

4.9 Host/DMA Port Priority

There is only one pair of request and grant signals on the Host/DMA Port. Since there can be three sources trying to access the Host/DMA Port, the priority is set as follows:

1. Serial Request from Serial Interface
2. APU Read
3. APU Write
4. DMA Operations (Read or Write)

When the APU attempts a load, and the Host/DMA Port is still busy with a DMA operation, the APU load preempts the DMA operation (APU steal cycle). The Host/DMA Port asserts the DMA Operation Ending signal, $\overline{\text{HBS\_END}}$, to suspend the DMA while performing the load. The Host/DMA Port asserts a new Host/DMA Port Address Strobe, HBS_AS, in the following cycle, along with the new address for the APU load operation. When the Host/DMA Port slave asserts the Host/DMA Port Read/Write Acknowledgment, $\overline{\text{HBS\_ACK}}$, to signal the end of the load transaction, the DMA operation resumes. The preempt mechanism also applies to APU stores. The DMA operations may be preempted more than once. However, when fast-page-mode DRAM is used, the preempt mechanism should be avoided. Software can avoid the preempt mechanism by not performing APU transactions while DMA is still busy.

4.10 Congestion Control

The ATMizer Architecture is capable of executing or facilitating almost any congestion control algorithm. The APU looks at the appropriate ATM Header Fields of each incoming cell for notification of congestion. If

congestion notification is found, the APU can take immediate action. Such actions may include one or more of the following:

1. Notify the Host that congestion has been seen utilizing the ATMizer Architecture-to-Host messaging scheme developed by the user
2. Lower the segmentation rate for each VC
3. Reduce the overall assigned cell throughput rate by setting a lesser value in the Global Pacing Rate Register
4. Set the CLP fields of outgoing cells to zero instead of lowering the overall information rate

4.11 APU External Access

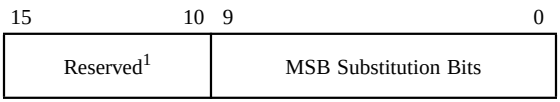
The APU can directly access the external devices through either the Host/DMA Port or the Secondary Port. For APU direct access loads and stores, the address space on both interfaces is 4 Mbytes. The APU has a four-word write buffer which enhances throughput during back-to-back store transactions. The APU also supports load scheduling, so a load operation does not stall the APU if the data is not required immediately.

Effective Address Bits [23:22] define whether the operation is a Host/DMA Port direct access or a Secondary Port direct access (see Figures 4.3 through 4.6).

This section explains APU direct access through the Host/DMA Port. Direct access through the Secondary Port is explained in Chapter 10.

Figure 4.1 shows the Host/DMA Port MSB Substitution Register format.

Figure 4.1 MSB Substitution Register



1. All reserved bits must be 0.

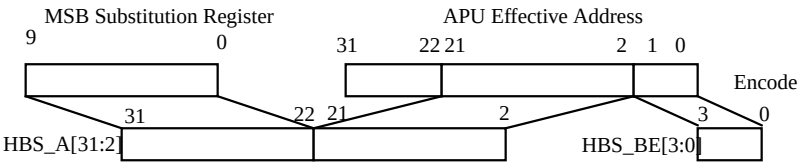
The MSB Substitution Register's Effective Address (the address used by the code) is 0xFFF04D00.

The MSB (Most Significant Bit) Substitution Register holds the 10 most significant bits of the address for an APU direct access through the Host/DMA Port. These 10 bits are valid only during an APU load or store through the Host/DMA Bus.

When the APU accesses the Host/DMA Port, it uses the 22 lower bits of the APU Effective Address and the 10 bits from the MSB Substitution Register to form the Host/DMA Port Address, HBS_A[31:2], and the Host/DMA Port Byte Enables, HBS_BE[3:0].

Figure 4.2 shows how the Host/DMA Port Address and the Host/DMA Port Byte Enables are formed from the APU Effective Address and the MSB Substitution Register.

Figure 4.2 Host/DMA Port Address and Byte Enables Formation



Figures 4.3 and 4.4 show the format of the Effective Address for APU direct access through the Host/DMA Port.

Figure 4.3 Effective Address for APU DMA/Host Port Access – Cacheable

31	30	29	28	27	26	25	24	23	22	21 … 0
0	0	0	0	0	0	0	A[1]	1	0	4-Mbyte Address Space

1. A =1 for atomic operation. A = 0 for single load/store.

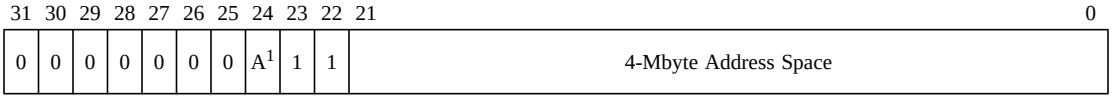
Figure 4.4 Effective Address for APU DMA/Host Port Access – Non-cacheable

31	30	29	28	27	26	25	24	23	22	21 … 0
1	0	1	0	0	0	0	A[1]	1	0	4-Mbyte Address Space

1. A =1 for atomic operation. A = 0 for single load/store.

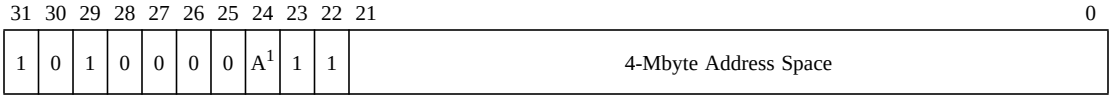
Figures 4.5 and 4.6 show the format of the Effective Address for APU direct access through the Secondary Port. Only 22 bits are required for SP access. Bits [21:0] of the APU Effective Address are transferred directly to SP_AD[21:0] during the address phase.

Figure 4.5
Effective Address for APU Secondary Port Access – Cacheable



1. A =1 for atomic operation. A = 0 for single load/store.

Figure 4.6
Effective Address for APU Secondary Port Access – Non-cacheable



1. A =1 for atomic operation. A = 0 for single load/store.

4.12 Prefetch Buffer

The Prefetch Buffer is a four-word data cache that works just like a normal write-through cache system. The ATMizer Architecture uses the Prefetch Buffer to efficiently load CPEs, buffer lists, and other data structures using burst or block fetches.

Whenever the APU accesses external memory devices with cacheable addresses, either port (Host/DMA Port or Secondary Port) can perform block fetches, and store the data in the Prefetch Buffer, enhancing system performance. The Block Fetch Size must be set to either two or four in both the System Control Register and the BIU/Cache Configuration (BCC) Register (refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual*).

When executing a block fetch from the SP, asserting SP_AD25 informs the memory system that the ATMizer Architecture is requesting a block on the Secondary Port. If the transfer is through the Host/DMA Port, the Host/DMA Port protocol handles the burst transaction automatically, which means that a four-word block fetch functions the same as a 16-byte word-aligned DMA operation.

The block fetch address is word aligned. The address wraps around, based on a two-word or a four-word boundary, as shown in Tables 4.1 and 4.2.

Table 4.1 Two-word Block Fetch Address Offset

Starting Address Offset	***Subsequent Address Offset***
0x0	0x4
0x4	0x8
0x8	0xC
0xC	0x0

Table 4.2 Four-word Block Fetch Address Offset

Starting Address Offset	***Subsequent Address Offset***	***Subsequent Address Offset***	***Subsequent Address Offset***
0x0	0x4	0x8	0xC
0x4	0x8	0xC	0x0
0x8	0xC	0x0	0x4
0xC	0x0	0x4	0x8

Chapter 5
Instruction RAM (IRAM) and Serial Interface

This chapter describes the function and operation of the Serial Interface and how to load the Instruction RAM.

This chapter has five sections:

5.1 Overview

The Instruction RAM (IRAM) is a 4-Kbyte single-cycle SRAM contained within the ATMizer Architecture. The IRAM holds up to 1024 instructions of user-written firmware that power the APU. During normal operation (when the APU is executing code) the APU can only read the IRAM. The APU can write to the IRAM during diagnostic mode.

To load the user firmware into the IRAM, the user has to disable the cache mechanism as explained in Section 5.5, “Loading Code into the IRAM Example Software.” Disabling the cache mechanism puts the IRAM into diagnostic mode. When the Serial Interface is enabled, the ATMizer Architecture stores the bitstreams from the Serial Interface or external logic into the Host/DMA Port or the Secondary Port memory. The user must then copy the user firmware into the IRAM using the Load IRAM Program in Section 5.5, “Loading Code into the IRAM Example Software.”

5.2 Serial Downloading

Serial downloading is the process of downloading code from a serial device through the Serial Interface to either the Secondary Port or the Host/DMA Port. The $\overline{\text{RST}}$, $\overline{\text{SRL\_BOOT}}$, HBS_BOOT, SRL_CLK16, SRL_ACK, and SRL_DIN signals control serial downloading.

To enable the serial downloading logic within the ATMizer, external logic must first assert $\overline{\text{RST}}$ LOW for at least four clock cycles. External logic must then deassert $\overline{\text{RST}}$ HIGH and keep $\overline{\text{SRL\_BOOT}}$ asserted LOW throughout the entire downloading process. When using a serial PROM, $\overline{\text{RST}}$ should be connected through an inverter to the PROM Output Enable signal ($\overline{\text{OE}}$) and SRL_ACK should be tied HIGH so that when the external logic deasserts $\overline{\text{RST}}$, it also enables the Serial PROM.

After external logic deasserts $\overline{\text{RST}}$, the ATMizer Architecture starts SRL_CLK16 (CLK divided by 16). Since the ATMizer Architecture CLK can be up to 50 MHz and Serial PROMs operate at less than 5 MHz, it is necessary to divide the clock by 16. SRL_CLK16 should be connected to the Serial PROM clock input. The Serial PROM presents data on SRL_DIN signal, and the ATMizer Architecture latches this data on the rising edge of SRL_CLK16.

During a serial download, the ATMizer Architecture takes data from the Serial Interface (SRL_DIN) one bit a time, packs the data into a 32-bit word and stores the data into either the Host/DMA Port or the Secondary Port depending on the state of the HBS_BOOT signal. Asserting HBS_BOOT selects the Host/DMA Port. Deasserting HBS_BOOT selects the Secondary Port.

The external system can control the rate of the serial data transfer using the SRL_ACK signal, since data on SRL_DIN is latched on the rising edge of SRL_CLK16 only when SRL_ACK is asserted. When the external system is not ready to present data on SRL_DIN, it can stall the ATMizer Architecture by deasserting SRL_ACK LOW. Whenever the external logic is able to provide data, it can assert SRL_ACK HIGH, which causes the ATMizer Architecture to receive more data. The ATMizer Architecture expects the serial bitstream to start with Bit 31 of Word 0 and continue to Bit 0 of Word N, where N is less than or equal to 4095. The bitstream fed into the ATMizer Architecture is not stored directly into the IRAM. Instead the bitstream is passed from the Serial Downloading Control Module to either the Secondary Port (if HBS_BOOT deasserted) or the Host/DMA

Port (if HBS_BOOT asserted) as 32-bit words. The ATMizer Architecture stores these instructions temporarily into system memory.

After the entire code (less than or equal to 4 Kbytes) has been stored into the system memory through the ATMizer Architecture, the external logic must deassert $\overline{\text{SRL\_BOOT}}$ HIGH to initiate the boot. Since it may be difficult to design logic that can deassert $\overline{\text{SRL\_BOOT}}$ exactly one cycle after the last bit of code has been loaded, LSI Logic recommends that the system designer create a RC Circuit to deassert $\overline{\text{SRL\_BOOT}}$ as soon as possible after the entire code has been loaded. This procedure may cause some extra data to be loaded into the Secondary Port or the Host/DMA Port, but this should not present a problem as long as the user program does not try to access this extra data.

5.3 Serial Interface Data Addressing

The ATMizer Architecture loads up to 4 Kwords in one download cycle. To provide flexibility for serial downloading, a portion of the first word that the Serial Interface fetches (the Segment Address) is used as the upper address for the serial downloading write operation. For a Host/DMA Port write operation, Segment Address Bits [31:14] are passed to The Host/DMA Port as HBS_A[31:14]. For a Secondary Port write, since there are only 22 bits in the address, Segment Address Bits [21:14] are passed to the Secondary Port as SP_AD[21:14]. Bits [31:23] and [13:0] of the Segment Address are ignored and should be cleared to zeroes. This format allows the system designer to dump the codes anywhere in the system memory on a 16-Kbyte boundary. As each word arrives, the ATMizer Architecture increments the Host/DMA or Secondary Port Address Bits [13:0] by four. Tables 5.1 and 5.2 show examples of how the Segment Address is used for the write address.

Table 5.1 Serial Interface Data Addressing through the Secondary Port

		Serial Download Write Address (SP_AD[21:0])			
Segment Address	***HBS_BOOT***	***Word 1***	***Word 2***	***. . .***	***Word 4095***
0x00000000	LOW	0x000000	0x000004	. . .	0x003FFC
0x00040000	LOW	0x040000	0x040004	. . .	0x043FFC
0x003FC000	LOW	0x3FC000	0x3FC004	. . .	0x3FFFFC

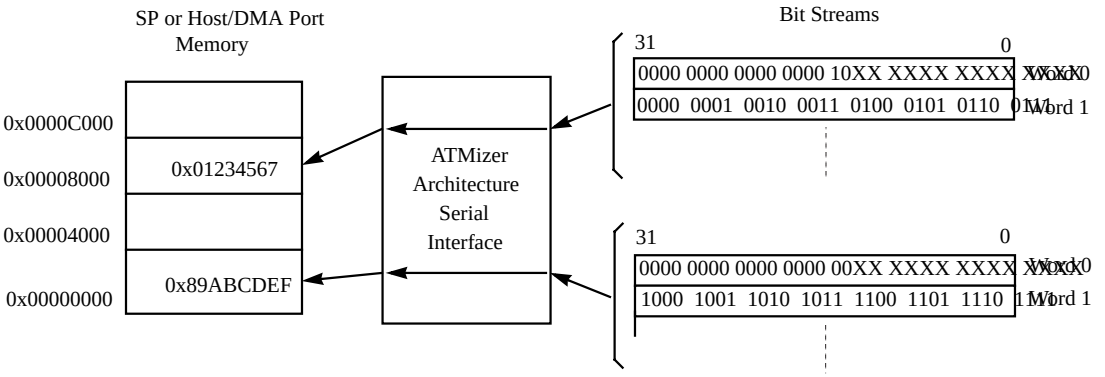
Table 5.2 Serial Interface Data Addressing through the Host/ DMA Port

		Serial Download Write Address (HBS_A[31:2] + Bits [1:0] = 00_2)			
Segment Address	***HBS_BOOT***	***Word 1***	***Word 2***	***. . .***	***Word 4095***
0x00000000	HIGH	0x00000000	0x00000004	. . .	0x00003FFC
0x01010000	HIGH	0x01010000	0x01010004	. . .	0x01013FFC
0xFFFFC000	HIGH	0xFFFFC000	0xFFFFC004	. . .	0xFFFFFFFC

When the code to be downloaded requires less than 4 Kwords, external logic can deassert $\overline{\text{SRL\_BOOT}}$ HIGH when downloading is finished. If the code to be downloaded requires more than 4 Kwords, external logic must deassert $\overline{\text{SRL\_BOOT}}$ at the 4-Kword boundary and at least 32 system clock (CLK) ticks later, assert it again (multiple downloading). The first word of the second download is decoded into another segment address the same as in the first download.

Figure 5.1 shows how the Serial Interface uses the serial input to address the data and store it.

Figure 5.1 Serial Downloading



5.4 Multiple Downloading and ATMizer Booting

The ATMizer Architecture allows the user to have code larger than 1 Kword. When the code for a particular application exceeds 1 Kword, non-timing critical routines may be kept in external, non-cacheable memory. The ATMizer Architecture can then execute this code.

To perform a second download, external logic must deassert $\overline{\text{SRL\_BOOT}}$ after performing the first download. This deassertion causes the ATMizer Architecture APU to fetch instructions from either the Host/DMA Port (if

HBS_BOOT is asserted HIGH) or the Secondary Port (if HBS_BOOT is deasserted LOW). After the second downloading process starts, the ATMizer Architecture arbitrates between the serial request and the APU Instruction Fetch operation. The serial request has the higher priority.

If, while performing the second serial downloading, the APU Host/DMA Port instruction fetch, a DMA operation, and a serial request all happen at the same time, the serial request has the highest priority, the APU has the second highest priority, and the DMA Engine has the lowest priority.

During a system reset, the APU hardware sets the APU Reset Vector to virtual address 0xBFC00000. The ATMizer Architecture maps this virtual address to a physical address based on the state of HBS_BOOT. Tying HBS_BOOT HIGH causes the APU, upon reset, to map the Reset Exception Vector virtual address to physical address 0x00000000 of the Host/DMA Port. Tying HBS_BOOT LOW causes the APU, upon reset, to map the Reset Exception Vector virtual address to physical address 0x00000000 of the Secondary Port.

The code at this boot starting address should have a jump instruction which transfers the APU program counter into the appropriate virtual address space (0xA0800000 to 0xA08FFFFC for the Host/DMA Port, or 0xA0C00000 to 0xA0CFFFFC for the Secondary Port).

Using the Secondary Port to boot, for example, the following instructions should be placed at the beginning of the memory location 0xA0C00000:

```
la r1, 0xA0C00500
j r1
```

The actual boot code starts at the address pointed to by `r1` (0x500 in this case). The jump ensures that the APU program counter generates the same effective address as the physical address. (Note that the jump must be an absolute jump as shown above.) From this point on, the APU program counter is now in the 0xA0C00000 to 0xA0CFFFFC range.

The BEV Bit (Bit 22) in the APU Status Register determines the APU General Exception Vector Address. A system reset sets the BEV Bit to one. Software in the APU may clear or set the BEV Bit.

Setting the BEV Bit to one sets the APU General Exception Vector Address to be the non-cacheable virtual address 0xBFC00180. The ATMizer Architecture maps this virtual address to a physical address

based on the state of HBS_BOOT. Tying HBS_BOOT HIGH causes the APU, upon a general exception, to map the General Exception Vector virtual address to physical address 0x00000180 of the Host/DMA Port. Tying HBS_BOOT LOW causes the APU, upon general exception, to map the General Exception Vector virtual address to physical address 0x00000180 of the Secondary Port.

LSI Logic recommends that the first instruction of the exception handlers (located at 0x00000180 on either the Host/DMA Port or the Secondary Port) redirect the APU using a jump instruction which transfers the APU program counter into the appropriate virtual address space (0xA0800000 to 0xA08FFFFC for the Host/DMA Port, or 0xA0C00000 to 0xA0CFFFFC for the Secondary Port).

Clearing the BEV Bit to zero sets the APU General Exception Vector Address to be the cacheable virtual address 0x80000080. The ATMizer Architecture maps this address to IRAM location 0x00000080 and upon a general exception jumps to this location.

After all the code is loaded into the IRAM through a series of load, invalidate, store, and validate operations, the APU should clear the IsC Bit to Validate the IRAM and then branch to the beginning of the executable code in the IRAM. This branch is a simple branch into a cacheable space as shown below (the beginning address of the IRAM, most likely at Address 0x0000000).

```
la r1, 0x00000000
j r1
```

5.5 Loading Code into the IRAM Example Software

In order to load instruction code from the Secondary Port or the Host/DMA Port into the IRAM, the APU must be programmed to invalidate the IRAM by setting the IsC Bit (Bit 16) in the APU Status Register.

In the example code below, the instructions to be loaded into the IRAM reside in the Secondary Port (SP) beginning at Address 0xA0C01000. There are 50 instructions to be loaded into the IRAM space beginning at Address 0x00000000.

In order to transfer 1024 instructions, the programmer must change the contents of `r5` from 50 to 1024. In order to load instructions from the Host/DMA Port into the IRAM, the programmer must change the Secondary Port Beginning Address into the Host/DMA Port Beginning Address. For

more detail explanation on the APU Internal Registers please refer to Chapter 14.

```
/*********************************************************
Function:     This program is used to load a test program into
              the IRAM from the Secondary Port (SP).
The program to be loaded into the IRAM is loaded at 0x1000 in
the Secondary Port memory.
*********************************************************/
#include “regdef.h”

.text
.set   noreorder
.set   noat

li r1, 0x00034800    # Value to be loaded into APU BCC Reg
                     # DS = 0
                     # IS1 = 1
                     # IBLKSZ = 0
                     # INTP = 0, NOPAD = 0, BGNT = 1
                     # LDSCH = 1
                     # NOSTR = 1
                     # (see
                     # Section 14.2, “APU Core Registers”)
la r10, 0xfffe0130   # APU BIU/Cache Config Reg Address
sw r1, (r10)         # Write to Config Reg
/*********************************************************
Program loop to load test program from the SP to the IRAM.
r5 contains the number of instructions to be transferred from
the SP to the IRAM.
In each pass of the loop, one instruction word is transferred
from the SP to the APU Register, and from there to the IRAM.
*********************************************************/
li r5, 50            # No. of instr words to fetch from SP
li r2, 0x00010000    # Bit 16 = 1 -> isolate cache
la r11, 0xa0c01000   # SP address (physical address 0x1000)
                     # Where the IRAM program to be transferred
                     # resides
la r12, 0x0          # The first location in the IRAM
loop: lw r3, (r11)   # Fetch IRAM instr word from SP

addi r11, 0x4        # Point to next IRAM instr word in SP
```

```
mtc0 r2, $12        # Write to CP0 Status Reg, enable
                    # Isolate cache
sub r5, 1           # Decrement counter by 1

sw r3, (r12)        # store instr word from SP to IRAM

mtc0 r0, $12        # write to CP0 Status Reg, disable isolate
                    # cache
bne r5, r0, loop    # fetch next IRAM instruction word
addi   r12, 0x4     # point to next addr in IRAM

nop                 #
nop                 #
nop                 #
j r0,               # r0 contains value “0” which is the
                    # starting address of the IRAM
nop                 #
nop                 #

# end
```

Chapter 6
Virtual Channel RAM (VCR)

This chapter describes the function and operation of the Virtual Channel RAM.

This chapter has five sections:

6.1 Overview

The Virtual Channel RAM (VCR) is a 1024 x 32 dual-ported Read/Write SRAM that provides the ATMizer Architecture with many of its unique capabilities. Almost all ATMizer Architecture operations involve the transfer of data to and from the VCR. The VCR can be read and written by the DMA Controller, the ATM Cell Interface and the APU.

All incoming cells (cells arriving over the ACI Receiver) are written into the VCR prior to processing. The APU decides whether to terminate a cell (reassemble it into a CS-PDU or a data buffer) or to switch a cell (internally or externally). All outgoing cells are either constructed in the VCR (segmentation) or transferred to the VCR (external switching) prior to transmission. In addition, Channel Parameter Entries (CPEs), memory buffer lists, messages, and other parameters can all be stored within the VCR. This ability to store parameters inside the ATMizer Architecture allows the ATMizer Architecture to be used in a variety of cost-sensitive applications such as memory-less network interface cards that support a limited number of simultaneously active VCs.

In high-end applications, it is possible to support an unlimited number of simultaneously active transmit and receive channels by storing all CPEs

externally. This procedure puts certain demands on the speed of local memory that may force the usage of SRAM for CPE storage.

The VCR is the most configurable aspect of the ATMizer Architecture. The software partitioning of the VCR can vary dramatically between applications. The VCR configuration affects the number of channels supported and the size, structure, and speed of the external memory system.

All cells received from the ATM Cell Interface are written into the VCR to await either reassembly or switching operations initiated by the APU. AAL 1, 2, 3/4, and 5 cells are built in the VCR by a combination of DMA operations and APU operations before being passed to the ACI Transmitter. The VCR may also be used to store CPEs, available buffer lists, and other data structures required for system operation. Some applications store CPEs in the VCR, while other applications store CPEs in main memory. Some applications may store CPEs in both places.

The VCR can be used for the following functions:

- Storing Cells
- Storing Channel Parameter Entries (CPEs)
- Cell Multiplexing and Demultiplexing

The following three sections describe these functions in more detail.

6.2 Storing Cells

The VCR can be used to store incoming and outgoing cells.

Incoming Cells

The Receiver in the ATMizer Architecture ATM Cell Interface reconstructs cells received from the external transmission convergence framing logic in the VCR. The ACI allocates 64 bytes of VCR memory to each incoming cell, although the actual cell size is user selectable (up to 64 bytes). The cell size must be programmed into the System Control Register as part of the APU's system initialization routine. The first 256 bytes (4 cells), 512 bytes (8 cells), 1024 bytes (16 cells) or 2048 bytes (32 cells) of the VCR are set aside for Received Cell Holders (selected by the BUFSIZ Field in the System Control Register). Cells are written into the VCR modulo 4, 8, 16, or 32. Cells must be processed before they are overwritten. Cell buffering in the VCR helps to decouple the incoming cell stream from memory interface latency and is especially helpful in situations where the

APU is temporarily unable to process incoming cells due to execution of an extended routine.

Cells written into the VCR are processed in the order of their arrival by the APU and are either switched over the internal Transmitter, switched over the main memory interface, or reassembled into memory-based realtime data stream buffers or CS-PDUs. The decision to switch or terminate a cell is made by the APU after examining the information stored in the CPE for the VC over which the cell arrived.

Outgoing Cells

All cells must be either moved to (external switching) or constructed in (segmentation) the VCR prior to transmission. Firmware can set aside an area in the VCR to act as the staging area for cell switching and generation. Outgoing cells are transferred from the VCR to the external transmission convergence framing logic by the ACI Transmitter. The ACI Transmitter uses VCR memory pointers. Whenever the APU wishes to have a VCR resident cell transferred to the Transmission Convergence Framing Logic, it simply writes a VCR pointer to the cell into the ACI Transmitter Cell Address FIFO. The ACI Transmitter then performs the transfer.

This pointer method puts no restrictions on the internal location of cells slated for transmission except that they be in the VCR. The ATMizer Architecture can switch Received Cell Holder resident cells out over the Transmitter by simply passing a pointer (to the cell) to the Cell Address FIFO (internal switching). To switch a cell from an external device (to source a pre-existing memory based cell out over the ATMizer Architecture ACI Transmitter), the APU must first initiate a DMA operation to bring the cell into the VCR from some temporary memory buffer. Once the cell is in the VCR, the APU passes the VCR pointer for the cell to the Cell Address FIFO in the same as it does for internal switching. Segmentation requires ATM and SAR (AAL 1, 2, and 3/4) Headers and Trailers (AAL 2 and 3/4) to be appended to the SAR SDUs by the APU. Once a cell is constructed in the VCR, the APU again passes a cell pointer to the Cell Address FIFO. The ACI Transmitter sends the cell to the transmission convergence framing logic, one byte at a time.

6.3 Storing Channel Parameter Entries (CPEs)

The VCR can be used for storing Channel Parameter Entries.

Channel Parameter Entries

A Channel Parameter Entry contains all of the pertinent information about a single VC. A system that supports AAL 5 CS-PDU segmentation and reassembly requires less information in a CPE than a system that supports AAL 5 CS-PDU segmentation and reassembly and cell switching. A system that supports simultaneous segmentation and reassembly of AAL 1, 2, 3/4, and 5 CS-PDUs requires an even more robust CPE for each VC.

The ATMizer Architecture does not enforce any specific CPE format. The structure and location of CPEs are defined by the user. User firmware dictates the CPE format, how VCs are grouped together, and how the segmentation process is performed on a grouping. The system designer creates the CPE format to fit his system and then writes the APU firmware to work within this environment.

For the APU to generate a cell, it must know certain information about the Virtual Circuit (VC) over which the cell will pass and information about the CS-PDU from which the cell is generated. This information includes:

1. The main memory address of the CS-PDU or realtime data buffer from which the SAR-SDU is retrieved
2. The number of bytes remaining in the CS-PDU or CS-PDU fragment (in scatter-gather applications)
3. In scatter-gather applications, whether or not the current CS-PDU fragment is the last fragment of a multi-fragment CS-PDU
4. The ATM Header that is to be appended to each cell
5. The ATM Adaptation Layer type (number) that is to be used to activate the appropriate CRC circuitry and to segment or reassemble cells originating or terminating on the given VC
6. The previous SAR Header/Sequence Number for AAL 1, 2, and AAL 3/4
7. The CRC32 Partial Result for AAL 5 CS-PDU

Collectively, these parameters provide the APU with all of the information that is needed to process an incoming cell or to segment a CS-PDU into a stream of cells.

Tables 6.1 and 6.2 show the CPEs implemented by LSI Logic for the ATMizer R/T System Development Platform Demo supporting AAL 5 CS-PDU segmentation and reassembly only.

Table 6.1 CPE for CS-PDU AAL 5 Segmentation

Entry	***Size***
CPE Link List Pointer to Next	4 Bytes
CPE Link List Pointer to Previous	4 Bytes
CS-PDU Memory Starting Address	4 Bytes
Base ATM Header (to be appended to each cell)[1]	4 Bytes
CRC32 Partial Result	4 Bytes
Next DMA Address	4 Bytes
Channel Group Number	4 Bytes
Number of Bytes Left and Total Byte Count[2]	4 Bytes

1. The APU handles PTI and CLP manipulation.
2. Two bytes for the number of bytes left and two bytes for the total byte count.

Table 6.2 CPE for CS-PDU AAL 5 Reassembly

Entry	***Size***
CS-PDU Memory Starting Address	4 Bytes
CRC32 Partial Result	4 Bytes
CRC32 Final Result	4 Bytes
Next DMA Address	4 Bytes

Channel Groups

A Channel Group is a group of VCs whose CPEs form a contiguous list, either in the VCR or in main memory. Channel Groups are defined by the user. The VCs that form a Channel Group reach their segmentation service intervals simultaneously (they are driven by a common PRPC). Once a PRPC times out, firmware running on the APU sequences through the list of VCs/CS-PDUs (the Channel Group), generating a specified number of cells from each CS-PDU before proceeding on to the next entry in the Channel Group (see Figure 6.1). The number of cells generated from each CS-PDU before proceeding to the next Channel Group entry (and therefore, the next CS-PDU) is controlled by user firmware.

Figure 6.1 shows some examples of software structures that can be stored in the VCR.

In the example shown in Figure 6.1, a CPE for a VC over which we are segmenting and transmitting a CS-PDU, requires 32 bytes of information. These 32 bytes include:

- Four bytes for the CPE link list pointer, which points to the next CPE
- Four bytes for the CPE link list pointer, which points to the previous CPE

- Four bytes for the starting address of a CS-PDU in the memory
- Four bytes for storing the ATM Header to be appended to each SAR-PDU
- Four bytes for CRC32 partial storage (AAL5)
- The next DMA starting address
- Channel group number used for the VC
- Two bytes for the number of bytes left, and two bytes for the total byte count for the CS-PDU

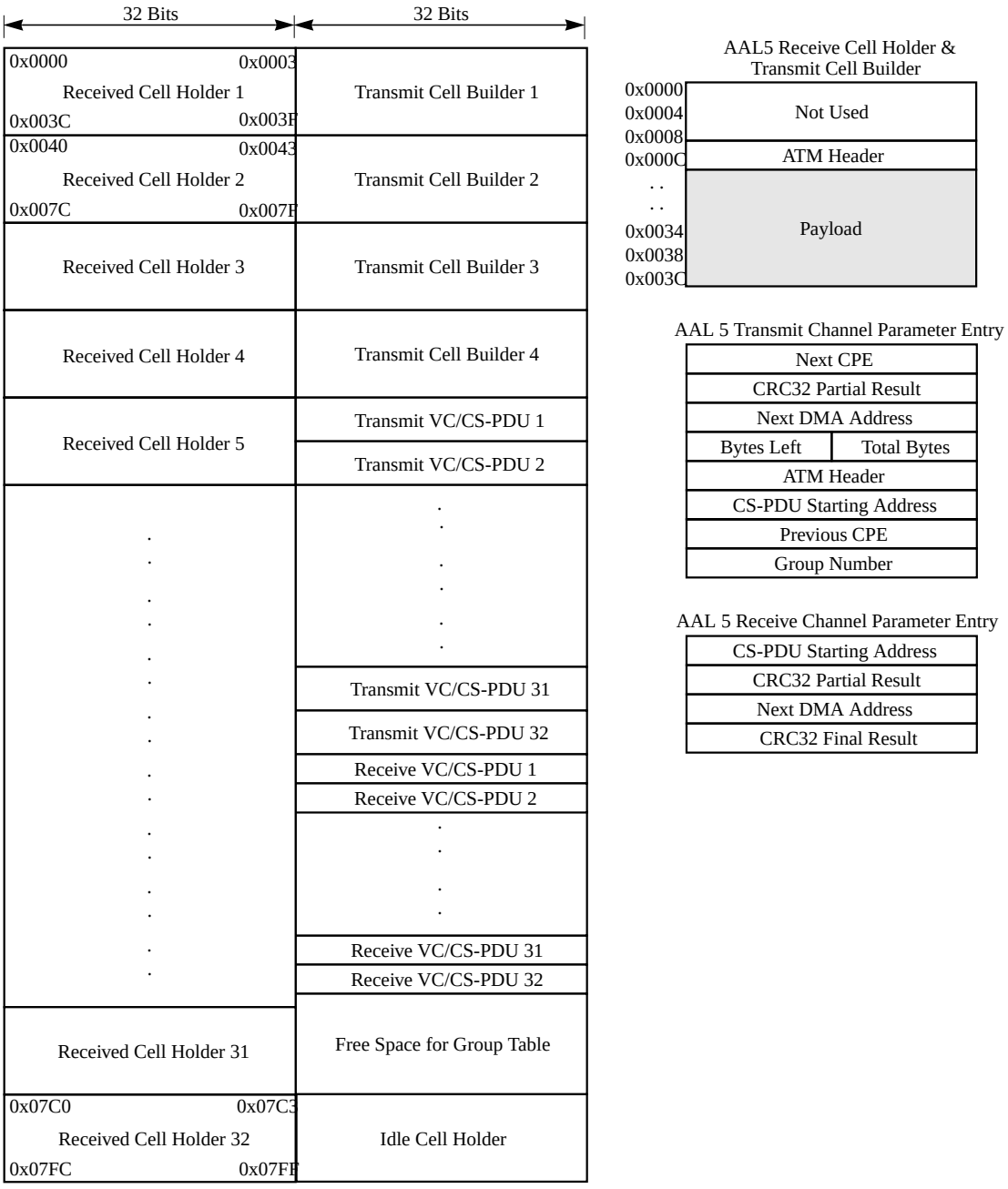
Note that a Channel Group is a user defined data-structure and not a structure enforced by the ATMizer Architecture.

The Host System manages CS-PDU sequencing over a single VC through either a linked-list mechanism (parsing driven by the ATMizer Architecture) or through an explicit messaging mechanism (the Host waits for a message from the ATMizer Architecture that indicates that CS-PDU Segmentation is complete, and then passes a new CS-PDU to the ATMizer Architecture to be segmented and transmitted over the VC).

Passing a new CS-PDU to the ATMizer Architecture means passing a new CPE to the ATMizer Architecture along with an indication of which Channel Group/PRPC to append the CPE. The ATMizer Architecture appends this new entry to the specified Channel Group. The Host uses memory mailboxes and Host-to-ATMizer messaging to pass a new CPE to the ATMizer Architecture. CPEs for channels carrying CS-PDUs undergoing reassembly can be built more compactly than CPEs for channels carrying CS-PDUs undergoing segmentation.

Figure 6.1 shows a possible VCR construction for a system supporting only 32 active VCs for both transmit and receive directions. In this example, the APU uses the VCI contained in the ATM Header of an incoming cell as an index into a look-up table to retrieve the CPE for the VC. CPEs for receiver-oriented channels are listed in order of their VCIs. This restriction does not apply to the transmit direction where a grouping and parsing mechanism is employed.

Figure 6.1
Example VCR Software
Structures



6.4 Cell Multiplexing and Demultiplexing

The ATMizer Architecture can simultaneously handle up to 65536 VCs, performing cell multiplexing and pacing for all of the active channels. However, there are trade-offs to be made between the number of channels supported, the data rate of the ATM Port, and the cost and structure of the memory.

For example, in a network interface card operating at desktop speeds less than or equal to 45 Mbits/s, it is possible to limit the number of VCs supported to 64 (32 Transmission and 32 Receive). In this case:

- The VCR can be used to cache all the relative parameters for each of these channels
- The ATMizer Architecture need only access main memory to retrieve and retire SAR-SDUs and Host Memory can be used for CS-PDU storage
- The NIC itself need not contain any memory

In applications requiring the support of a very large number of channels, the VCR can not hold all of the needed channel information. It may be necessary to provide high-speed SRAM, that can be accessed by the ATMizer Architecture Secondary Port, for Channel Parameter Entry storage. The SRAM gives the ATMizer Architecture fast access to the information needed for segmenting and reassembling CS-PDUs and for the switching of cells. CS-PDU storage could be handled in a local DRAM or SRAM-based memory system that is accessible by the ATMizer Architecture DMA Controller.

Systems can be a cross between the two examples above. In some systems, it is possible to limit the number of simultaneously active transmission channels. There is no limit on the number of transmission VCIs supported, only in the number that can have CS-PDUs under segmentation at any one time. If the number is limited to 32, then all Transmission Channel Parameters can be cached internally. The time saved by caching Transmission Parameters in the VCR can be used to retrieve the parameters needed for reassembly. This time usage may allow the use of a single interleaved DRAM system for both CS-PDU and Channel Parameter storage. An unlimited number of transmission VCIs can be supported by swapping out a CPE/VC/CS-PDU for a new CPE/VC/CS-PDU once its CS-PDU (or CS-PDU fragment) has been segmented.

6.5 VCR Partitioning Examples

Figures 6.2 and 6.3 show two examples of VCR constructions.

Figure 6.2 shows the VCR partitioning for a Network Interface Card (for a PC or a Workstation) that supports a limited number of open channels. In this example, all Channel Parameter Entries (CPEs) for both transmit and receive channels are stored in the VCR, eliminating the need for local memory in the Secondary Port.

Figure 6.3 shows the VCR partitioning for a router that supports an unlimited number of open channels but places a restriction on the number of VCs that can have CS-PDUs under active segmentation at any one time. This example system limits the number of transmission channels that can be active simultaneously to 64, and caches all the CPEs for active channels in the VCR. This example does not limit the number of open transmission channels, only the number of channels that can have CS-PDUs undergoing segmentation simultaneously.

Figure 6.2
VCR Partitioning for an NIC

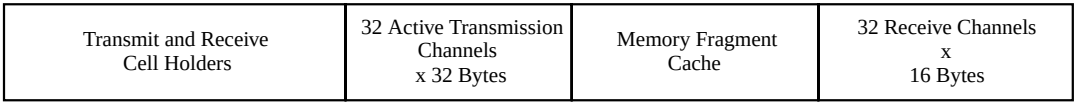


Figure 6.3
VCR Partitioning for a Router

Once one CS-PDU has been completely segmented, the APU can swap out its CPE for the next in line. CPEs for channels that are active in the receive direction are stored in external local memory, which allows the router to support an unlimited number of simultaneously active receive channels. Without an intelligent memory fragment allocation plan, support for a large number of VCs would overload most memory systems. Fortunately the ATMizer Architecture combines support for external CPEs with a capability to do link list-based CS-PDU scattering during reassembly allo-

cating memory in fragments as small as necessary. The net result is that the example router is able to support an unlimited number of open transmit and receive channels from a single unified DRAM-based memory system with a single restriction on the number of transmission channels that can be actively undergoing segmentation at one time.

Chapter 7
Pacing Rate Unit (PRU)

This chapter describes the function and operation of the Pacing Rate Unit.

This chapter has eight sections:

7.1 Overview

The Pacing Rate Unit (PRU) implements the Peak Rate Pacing and Maximum Burst Length control functions. When one of the 10 Peak Rate Pacing Counters (PRPCs) reaches zero, the firmware starts to segment cell(s) from all CS-PDUs associated with that PRPC. Anytime one or more PRPCs has timed-out but has not yet been serviced, internal hardware asserts the APU input CpCond2 or the Interrupt1 signal depending on the timeout mode selected in the Configuration Register. Firmware running on the APU periodically checks the state of CpCond2 by executing the Branch on Coprocessor Condition 2 True Instruction. If CpCond2 is true, at least one PRPC has timed-out and the APU must segment the CS-PDUs attached to any PRPCs that have reached their service intervals.

The APU determines which PRPCs have timed-out by reading the 10-bit Channel Group Credit Register (CGCR). Each bit set in the CGCR indicates that the corresponding PRPC has timed-out. The PRPC can be clocked by the System Clock (CLK) or the transmission line clock connected to the PRU_CLK pin. Note that the maximum frequency of

PRU_CLK cannot exceed one half of the System Clock frequency. In some applications, the PRPC counters must count with the transmission line clock, so the Transmission Clock, TX_CLK, is connected to PRU_CLK.

The Pacing Rate Unit consists of:

- Peak Rate Pacing Counters (PRPCs)
- Channel Group Credit Register (CGCR)
- Count Initialization Register (CIR)
- Configuration Register (CR)
- Stall Register (SR)

The following five sections describe these components.

Note The CpCond and Interrupt signals are internal to the APU core which is part of the ATMizer Architecture. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

7.2 Peak Rate Pacing Counters (PRPCs)

The APU can use the Peak Rate Pacing Counters for pacing the rate of CS-PDU segmentation, implementing the leaky bucket algorithm, controlling usage parameters, or as general purpose counters. Each counter has an associated Count Initialization Register (CIR). Eight of the PRPCs and their associated CIRs are 12 bits wide. Two of the PRPCs and their associated CIRs are 24 bits wide.

The 10 Peak Rate Pacing Counters can be used to control the rate of CS-PDU segmentation. Whenever one or more PRPCs times out, the PRU asserts the APU CpCond2 or Interrupt1 input to inform the APU firmware, which will then branch to the Segmentation routine.

The 24-bit PRPCs can be used to implement the leaky bucket algorithm. One counter tracks the rate to fill the bucket, and the second counter tracks the number of tokens in the bucket.

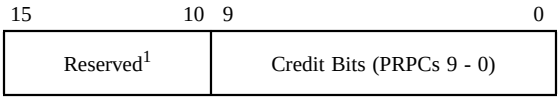
7.3 Channel Group Credit Register (CGCR)

The Channel Group Credit Register is a 16-bit APU read/write register containing one bit for each PRPC (PRPCs 9 through 0 directly correspond to Credit Bits [9:0]). When the PRPC counts down to zero, the PRU sets the corresponding PRPC Credit Bit in the CGCR. Software can read the contents of the CGCR at any time to see which PRPCs has timed-out. The

APU should use either the Load Halfword or Load Word instruction to read the CGCR. Reading the CGCR clears the CGCR.

Figure 7.1 shows the Channel Group Credit Register format.

Figure 7.1 Channel Group Credit Register



1. All reserved bits must be 0.

The CGCR's Effective Address (the address that is used by the code) is 0xFFF040X0.

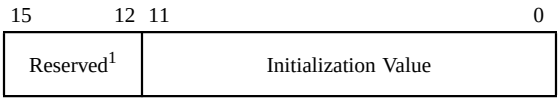
Since some PRPCs are associated with CpCond2 and other PRPCs are associated with interrupts, the PRU allows the APU to indicate which bits in the CGCR should be cleared when reading the register. Clearing the CGCR Effective Address Bit 7 to zero causes the PRU to clear only those bits configured for the CpCond2 method. Setting the Effective Address Bit 7 to one causes the PRU to clear only those bits configured for the Interrupt Method. The APU should clear Effective Address Bits [6:4] to zero. Firmware running on the APU can implement channel priority by selectively servicing Channel Groups that have timed-out.

7.4 Count Initialization Register (CIR)

The APU initializes the PRPC counter value by writing to the corresponding Count Initialization Register. For PRPCs 0 through 7, the initial value is placed on CIR Bits [11:0] and for PRPCs 8 and 9, the initial value is placed on CIR Bits [23:0]. The APU can also read the 10 PRPCs at the same addresses (the 10 addresses are read/write). Store Halfword instructions should be used to write to PRPCs 0 through 7, and Store Word instructions should be used to write to PRPC 8 and 9. Load Halfword instructions should be used to read PRPC 0 through 7 and Load Word instructions should be used to read PRPC 8 and 9.

Figure 7.2 shows the 12-bit Count Initialization Register format used for PRPCs 0 -7.

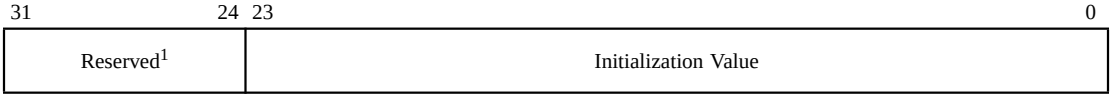
Figure 7.2 12-Bit Count Initialization Register



1. All reserved bits must be zero.

Figure 7.3 shows the 24-bit Count Initialization Register format used for PRPCs 8 -9.

Figure 7.3
24-Bit Count
Initialization Register



1. All reserved bits must be 0.

The Count Initialization Register's Effective Addresses are 0xFFF043XX (see Table 11.8).

Setting Effective Address Bit 7 to one causes the PRU to write the initialization value into the CIR and the PRPC immediately. Clearing Effective Address Bit 7 to zero causes the PRU to write the initialization value into the CIR immediately, and to update the PRPC after it reaches zero. Effective Address Bit 6 should be cleared to zero. Table 7.1 shows how the Effective Address Bits [5:0] select the CIR and the size of the CIR.

Table 7.1
Effective Address
Bits [5:0] CIR
Selection

Bits [5:0]	***CIR***	***Size (Bits)***
000000_2	0	12
000010_2	1	12
000100_2	2	12
000110_2	3	12
001000_2	4	12
001010_2	5	12
001100_2	6	12
001110_2	7	12
100000_2	8	24
100100_2	9	24

7.5 Configuration Register (CR)

The Configuration Register has Timeout and Clock Select fields for each corresponding PRPC.

The Timeout Field is used to notify the APU when a PRPC has counted down to zero. There are two methods for indicating timeout to the APU. In the first method, the APU can check to see if a PRPC has timed-out by executing a Branch On Coprocessor 2 Condition Instruction. The PRU asserts CpCond2 as long as at least one PRPC associated with the CpCond2 timeout indication method has its bit set in the CGCR. In the second method,

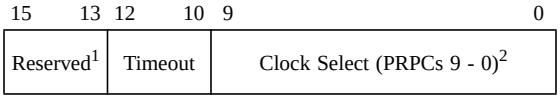
the PRU asserts a timeout interrupt. The PRU asserts Interrupt1 to the APU as long as at least one PRPC associated with the interrupt timeout indication method has its bit set in the CGCR. A PRPC may be configured to use either timeout indication method, but not both.

PRPCs 0, 1, 2, and 3 are grouped, PRPCs 4, 5, 6, and 7 are grouped, and PRPCs 8 and 9 are grouped. Software must determine which method each group of timers will use for timeout indication. Table 7.2 shows how the Timeout Field is used for this purpose. Software can change the timeout mechanism dynamically. When one of the counters in a group times-out, software must read the CGCR to find out which PRPC caused the CpCond2 or the interrupt.

Each PRPC can be driven by either the System Clock (CLK) or by the Pacing Rate Unit Clock (PRU_CLK). The APU uses the Clock Select Field to select between these two clocks. Bits [9:0] select the clock for PRPCs 9 through 0. Clearing the PRPC's corresponding bit to zero causes the PRPC to be clocked using CLK. Setting the PRPC's corresponding bit to one causes the PRPC to be clocked using PRU_CLK.

Figure 7.4 shows the Configuration Register format.

Figure 7.4 Configuration Register



1. All reserved bits must be 0.
2. CLK, if bit cleared to zero. PRU_CLK, if bit set to one.

The Configuration Register's Effective Address is 0xFFF04100.

Note

Firmware must *not* write to the Configuration Register using a Store Halfword instruction.

Table 7.2 PRPC Grouping

Timeout Method[1]	***PRPCs***
Bit 10	0, 1, 2, 3
Bit 11	4, 5, 6, 7
Bit 12	8, 9

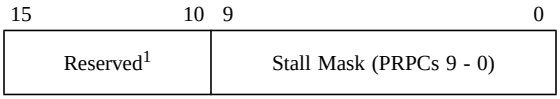
1. CpCond Method, if bit cleared to zero. Interrupt Method, if bit set to one.

7.6 Stall Register (SR)

The PRU contains a Stall Register used to suspend one or more PRPCs. Setting a Stall Mask Bit to one in the Stall Register causes the corresponding PRPC to stop counting. Stall Mask Bits [9:0] correspond to PRPCs 9 through 0. Credit does not accumulate for a PRPC with its Stall Mask Bit set to one or when a PRPC is stalled at zero. If a PRPC is stalled at zero, the PRU sets its Credit Bit as soon as the Stall Bit is cleared. Once the Stall Mask Bit is cleared to zero, the corresponding PRPC begins counting again from where it left off. The APU should use either the Store Halfword or the Store Word instruction to write to this register.

Figure 7.5 shows the Stall Register format.

Figure 7.5 Stall Register



1. All reserved bits must be 0.

The Stall Register's Effective Address is 0xFFF04200.

7.7 Cell Rate Pacing

The ATMizer Architecture can implement Cell Rate Pacing by controlling the CS-PDU Segmentation Rates.

Peak Rate Pacing and Burst Length

The ATMizer Architecture can implement the ATM Layer Peak Rate Pacing and Maximum Burst Length control functions.

Once a CS-PDU or CS-PDU fragment has been passed to the ATMizer Architecture for segmentation, the ATMizer Architecture controls the rate of cell generation from the CS-PDU and the number of back-to-back cells generated from each CS-PDU. A CS-PDU can be attached to any of the 10 user-programmable Peak Rate Pacing Counters in the ATMizer Architecture. Each PRPC counts down by one on each clock tick. Since each CS-PDU attached to a given PRPC may have its own Burst Length value, the count in the Peak Rate Pacing Register actually determines the Service Interval for the Channel Group, not necessarily the peak rate of cell generation for CS-PDUs attached to that PRPC. CS-PDUs attached to a particular PRPC with similar characteristics, such as channel priority, are collectively referred to as a Channel Group. More than one Channel Group can be attached to a single PRPC. If the Burst Lengths for each CS-PDU attached to a PRPC are identical, the PRPC count determines the actual peak rate of segmentation for CS-PDUs belonging to that Channel Group.

CS-PDUs are attached to a PRPC by the Host Processor. When the Host passes a Segment CS-PDU information packet to the ATMizer Architecture, it includes in the information packet an indication of which PRPC should be used to define the Service Interval for segmenting the CS-PDU. It also includes the Burst Length value for the CS-PDU (how many cells should be generated and sent, back-to-back, for the CS-PDU at each service interval). The ATMizer Architecture, upon receiving this Segment CS-PDU information packet (through Host-ATMizer Messaging) appends the Channel Parameters for the CS-PDU to the end of the specified Channel Group and begins the segmentation process on the CS-PDU the next time its associated PRPC times-out.

When servicing a Channel Group, APU firmware can generate and send one or more cells for one VC before servicing the next VC in the Channel Group. The number of cells to be sent before proceeding to the next Channel Group entry can be defined either by construction (this value is the same for each member of a Channel Group and embedded into the firmware directly) or by a field inside the Channel Parameter Entry for the VC. Firmware running on the ATMizer Architecture segments the number of cells specified by this Burst Length value before proceeding to the next Channel Group entry. A side effect of this process is that the amount of time required to access and restore a Channel Parameter Entry can be amortized over several cells, effectively reducing the number of APU instructions and the amount of time required to generate a cell. This time savings may be of importance in high-speed applications (155 Mbits/s) supporting a large number of VCs (more than 512).

Average Pacing

Average Pacing may not be implemented by the ATMizer Architecture. It will probably be implemented by the Host Processor, which has access to a realtime clock.

To maintain the Average Pacing Rate agreed to at connection establishment, the Host Processor keeps a running total of the number of bytes sent over each established VC. Prior to queuing a new CS-PDU for segmentation over a given VC, the Host Processor must first determine if queuing the CS-PDU would violate the Average Rate for the VC. To do this the processor calculates the amount of time that has passed since the last checkpoint. It then divides the total number of bytes sent out over the VC since the last checkpoint by the elapsed time. The result is the actual Average Pacing Rate in bytes per second. If queuing the next CS-PDU would result

in a violation of the agreed to Average Pacing Rate for the Virtual Circuit, then the Host Processor waits a period of time before passing the CS-PDU to the ATMizer Architecture for segmentation. If queuing the CS-PDU would not violate the Average Pacing Rate parameter, the CS-PDU is passed to the ATMizer Architecture for segmentation. As statistical multiplexing issues become better understood, software can be modified to implement Average Rate Pacing in the best way.

7.8 Channel Priority

Firmware can use the CGCR to implement virtually any Channel Priority algorithm. There are no priority mechanisms enforced in hardware. By checking the CGCR Bits in a particular order, the APU can implement high-priority and low-priority Channel Groups. To give higher priority to CS-PDUs/VCs belonging to high-priority Channel Groups, the APU can read the CGCR during the servicing of a lower-priority Channel Group to see if a higher-priority Channel Group has timed-out. If it has, the APU can suspend servicing the lower-priority Channel Group and service the higher-priority Channel Group. After servicing all higher-priority channels, the APU can resume servicing of the lower-priority channel where it left off.

The user can attach both high-priority and low-priority CS-PDUs to a single PRPC in order to pace high-priority and low-priority CS-PDUs/VCs at the same Service Interval Rate. Each PRPC can have two (or more) Channel Groups associated with it. A PRPC may have a high-priority Channel Group and a low-priority Channel Group attached to it. The APU can service all channels belonging to the high-priority Channel Group and then check for pending high-priority requests by reading the CGCR before servicing the low-priority Channel Group attached to that particular PRPC.

PRPCs and their associated Channel Group or Channel Groups can be given different priorities. If more than one Channel Group has reached its service interval and is awaiting servicing the following servicing protocol may be implemented:

- High-priority requests are serviced before low-priority requests
- Existing high-priority requests are serviced before new high-priority requests
- New high-priority requests are serviced before existing low-priority requests

This implementation of channel priority is different from the AAL 5 high-medium-low CS-PDU Priority assignment, but both of these priority constructions influence the cell generation process. Channel priority affects the Channel Group/CS-PDU servicing sequence. The ATM AAL 5 CS-PDU priority is reflected in the PTI and CLP fields of the ATM Header. Both functions are controlled by the ATMizer Architecture. For AAL 5 traffic, the Host must include the CS-PDU priority in the Segment CS-PDU message packet sent to the ATMizer Architecture.

Chapter 8
DMA Controller (DMAC)

This chapter describes the function and operation of the DMA Controller, which is contained within the Host/DMA Port.

This chapter has eight sections:

8.1 Overview

The APU uses the DMA Controller (DMAC) to perform data transfers between the VCR and Host/DMA Port external memory. The DMAC, which can perform block transfers up to 64 bytes, supports every combination of local and main memory byte alignment on transfers. This support for misaligned operations gives the ATMizer Architecture an ability to participate in robust Scatter-Gather operations. The DMA Controller includes registers, counters, and a data path that collectively control data transfer operations between the VCR and Host/DMA Port external memory.

The DMAC also generates CRC32 for AAL 5 CS-PDUs and can be used to retrieve and restore memory-based channel parameters.

The main DMA Controller functions include the following:

- Retrieve SAR user payloads from memory-based CS-PDUs during segmentation operations

- Write SAR user payloads back into memory-based CS-PDUs during reassembly operations
- Retrieve and restore application-specific data structures

The DMA Controller also contains CRC32-generation circuitry that generates the CRC32 values required for AAL 5 CS-PDU. CRC32 can be calculated individually for each CS-PDU actively undergoing either segmentation or reassembly. For CS-PDUs undergoing segmentation, the final CRC32 result is appended, under APU control, to Bytes [48:44] of the SAR-SDU of the last cell generated from the CS-PDU. For CS-PDUs undergoing reassembly, the CRC32 result is compared with the CRC32 received in the last cell of the CS-PDU as a checking mechanism. Because the ATMizer Architecture supports cell multiplexing and demultiplexing from up to 64K VCs, the APU must provide CRC32 partial result storage into CPEs and retrieval services to allow multiple concurrently active CRC32 calculations to be performed by the single CRC32 generator.

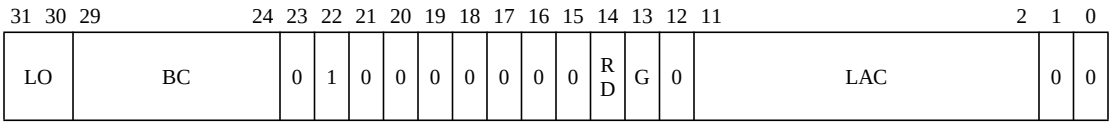
8.2 Registers

This section defines the registers used in the DMAC. Please note that some of the fields in the DMAC Control Register's Effective Address contain control data. Both the DMAC Control Register and some of the fields from the DMAC Control Register's Effective Address are used to configure the DMAC Registers and Counters. Using this method the user can program the DMA with one store instruction.

DMAC Control Register's Effective Address

Figure 8.1 shows the format of the Effective Address of the DMAC Control Register. For more information see Section 11.1, "Memory Maps."

Figure 8.1
DMAC Control Register's Effective Address



LO **Local Address Byte Offset** **[31:30]**
The APU firmware uses LO to inform the DMA Controller of the offset (in bytes) of the first byte of valid data in the VCR.

BC **Byte Count** **[29:24]**

The APU firmware uses BC to set the size (in bytes) of a DMA transfer. Since BC can only hold a six-bit value (from 0 to 63), a BC of 000000_2 sets the size to 64 bytes.

RD **Read/Write (Operation Direction)** **14**

This bit controls the direction of the DMA operation. The APU firmware sets this bit to one to indicate that it wishes to perform a read from main memory. The APU clears this bit to zero to indicate that it wishes to perform a write to main memory.

G **Ghost Bit** **13**

This bit is used to inform the DMAC that the DMA operation being programmed is being done solely for the purpose of creating a CRC32 Partial Result (an intermediate calculation) for an AAL 5 SAR-SDU that has been constructed in the VCR from two or more CS-PDU data block fragments. If a SAR-SDU is built from more than one data block, and if one of the data blocks was not word aligned and of size evenly divisible by four, the CRC32 Partial Generator in the DMAC is not able to calculate a correct CRC32 Partial Result for the SAR-SDU over the numerous DMA operations. Therefore, once the entire SAR-SDU has been built up in the VCR, the APU has to force a CRC32 partial generation by initiating a DMA Ghost Write operation.

When the Ghost Bit is set to one for a DMA write operation, the write transaction does not go out to the bus. The DMAC, however, performs the Ghost Write operation, which calculates the Partial Result and places it in the CRC32 Register. Once the operation is complete, the APU can read the Partial Result. The DMA Ghost Write operation is initiated by setting RD to zero and G to one in the Effective Address.

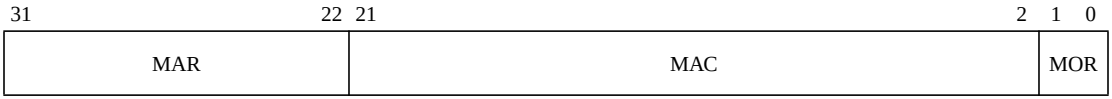
LAC **Local Address Counter** **[11:2]**

The Local Address Counter holds the VCR read or write word address (the local address). The APU initializes LAC with the Local Starting Address at the beginning of a DMA operation. The DMAC increments the LAC as the operation proceeds.

DMAC Control Register

Figure 8.2 shows the format of the DMAC Control Register.

Figure 8.2
DMAC Control Register



MAR **Memory Address Register** **[31:22]**

The Memory Address Register holds the ten most significant bits of the main memory address during DMA operations. While the DMAC does increment the main memory address for consecutive transfers in a multiple word DMA operation, it does not increment the value in the MAR. Therefore, if external logic relies on sequential incrementing of the Host/DMA Port Address Bus during multiple word DMA operations, the APU should not initiate a DMA operation that crosses a 16-megabyte boundary. The contents of MAR are reflected on output pins HBS_A[31:24] when the HBS_GNT signal is asserted.

MAC **Memory Address Counter** **[21:2]**

The Memory Address Counter holds the starting word address for the DMA operation. During a DMA operation, the Memory Address Counter is incremented when $\overline{\text{HBS\_ACK}}$ is asserted by external logic. The contents of the MAC are reflected on HBS_A[23:2] when HBS_GNT is asserted.

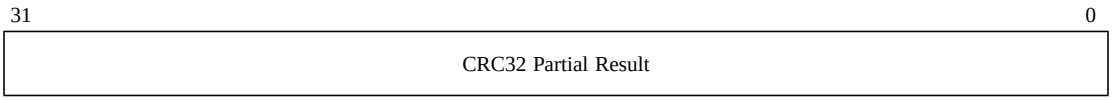
MOR **Memory (Byte) Offset Register** **[1:0]**

The Memory Offset Register holds the two least significant bits of the main memory starting address of a DMA operation. The DMAC starts the memory access beginning at the byte pointed to by MOR. The $\overline{\text{HBS\_BE}}$[3:0] outputs indicate which bytes should be stored to memory during DMA transfers.

CRC32 Register

Figure 8.3 shows the format of the CRC32 Register.

Figure 8.3
CRC32 Register



The CRC32 Register should be initialized to all ones by the APU prior to beginning the first SAR User Payload retrieval for an AAL 5 CS-PDU. The CRC32 Partial Result is an intermediate CRC32 calculation.

The CRC32 Partial Result, generated during the DMA operation, is read from the CRC32 Register by the APU at the end of the DMA operation. It is saved and then restored prior to the next segmentation operation. The register is also used this way for CRC32 generation during reassembly.

The CRC32 Register's Effective Address (EA[31:0]) is 0xFFF04CX0. Clearing EA7 to zero causes the DMAC to return the CRC32 Partial Result. Setting EA7 to one causes the DMAC to return the CRC32 Final Results. EA[6:4] must always be cleared to zeroes.

8.3 Programming the DMAC

In order to initiate a DMA operation between main memory and the VCR, the APU programs the DMAC with the starting main memory address (byte address), the local/VCR starting address (word aligned address written into the Effective Address LAC and the starting byte offset within the targeted word written into the Effective Address LO), the number of bytes to be transferred, and the direction of the transfer. In addition, the APU may need to preset the CRC32 Generator for AAL 5 CS-PDU CRC32 support or set the Ghost Bit.

Both the DMAC Control Register and some of the fields from the DMAC Control Register's Effective Address are used to configure the DMAC Registers and Counters. The APU can configure the DMAC Control Registers and Counters, and initiate a DMA operation by executing a single Store Word instruction. For AAL 5 CS-PDU Segmentation and Reassembly, if the ATMizer Architecture is to be used for CRC32 generation and checking, a second Store Word instruction is needed to initialize the CRC32 Generator with the correct CRC32 Partial Result value. This second instruction should be executed immediately before the Store Word

instruction that is used to initialize the DMAC Registers and initiate the DMA operation. The CRC32 Register can be read at the end of a DMA operation using a Load Word instruction.

8.4 Cell Switching, Segmentation, and Reassembly

The ATMizer Architecture, under APU user firmware control, can be used to implement CS-PDU segmentation, CS-PDU reassembly, and ATM cell switching. For each VC, the APU can decide whether to switch or terminate an incoming cell. The decision can be based on static principles (certain VC numbers can be dedicated to switched VCs while other VC numbers are dedicated to terminating VCs) or on dynamic principles (the CPE for a given VC can have a flag that indicates whether its cells should be switched or terminated).

Reassembly and Cell Switching

If an incoming cell is to be switched, it can be passed, headers and trailers intact, to any memory-mapped device using the ATMizer Architecture DMA Controller. In networks implementing a ringlike structure or a simple two-way switching matrix, incoming cells can be switched directly between the ACI Receiver and ACI Transmitter by simply passing a pointer (to the cell in the VCR - the cell's VCR starting address) to the ACI Transmitter (the same procedure that is used for queuing a cell for transmission). Using this method, cells can be switched within the ATMizer Architecture, without using system memory.

The APU may perform operations, such as Virtual Path Indicator (VPI)/ Virtual Channel Indicator (VCI) Translation, and Congestion Notification Insertion, on a cell before switching it. The APU performs these operations by overwriting new values into specific fields in the cell. For example, if VCI translation is required, the APU firmware sets a flag in the CPE, for the VC that received the cell, that indicates that the cell is to be switched with VCI Translation. The new VCI is included in the CPE as well. The APU reads the new VCI from the CPE and writes it into the VCI Field of the cell held in the VCR (the VCR holds either 4, 8, 16, or 32 64-byte cells and the ACI Receiver writes cells into the VCR using modulo 4, 8, 16 or 32). The APU firmware decides whether to switch the cell over the backplane using the DMA Controller or to pass a pointer to the cell to the ACI Transmitter. The specific procedures for implementing cell switching are always defined by user firmware.

From the perspective of the DMA Controller and the ATM Cell Interface, there is no distinction between cell switching and circuit termination. Cells

arriving over the ACI Receiver are written into the VCR. In the case of circuit termination, the APU initiates a DMA operation to transfer the User Payload portion of a cell to its corresponding memory-based CS-PDU and sets the LAC, LO, and BC values in the DMAC accordingly. In cell switching applications where a cell is to be transferred to a memory-mapped device, the entire cell (headers and trailers included) must be transferred. So the pointer written into the LAC should point to the beginning of the cell instead of the beginning of the SAR User Payload Field. The Local Offset is most likely zero, and the BC value should be large enough to include all switching information, ATM, and SAR headers and trailers.

The diagrams in Figure 8.4 (on page 8-9) show the local address pointers (labeled B) that would be written into the DMAC Local Address Counter, Local Offset Register, and Transfer Length Counter to perform Reassembly on 52- and 60-byte cells as well as the pointers (labeled A) that would be written into these same registers to perform Switching operations on 52- and 60-byte cells. The diagrams also point out that in the case of AAL 3/4 cells, the SAR User Payload is not word aligned in the VCR. Therefore, the APU must set the Local Offset Field to 10 when initiating the DMA transfer to inform the DMA Controller of the alignment condition. The DMA Controller merges bytes from two VCR words into a single word to be written to a word aligned data structure in main memory. If the MOR Bit indicates that the targeted memory address is not word aligned, the DMA Controller also adjusts the targeted local data to the proper memory alignment.

The DMAC has the capability to transfer from any local offset to any memory offset and vice versa. This capability is especially important in AAL 3/4 Segmentation and Reassembly operations, in AAL 3/4 and AAL 5 Gather operations, and in AAL 3/4 or AAL 5 Scatter operations where the system designer wishes to rely on the ATMizer Architecture to do higher layer (TCP/IP) header stripping and packet alignment to accelerate Application Layer routines.

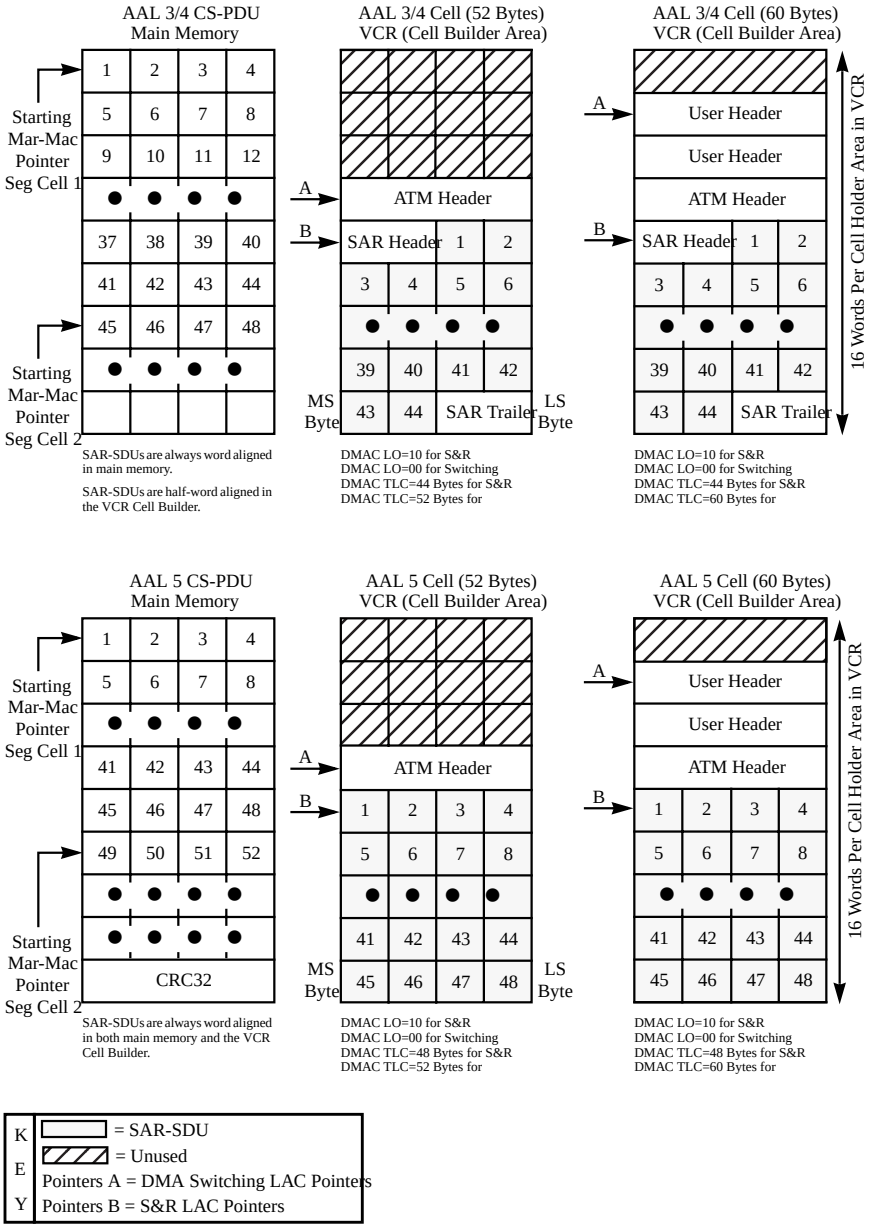
Note When switching AAL 3/4 Cells, the Local Offset should be set to 00_2 because even though the SAR User Payload Field is misaligned, the cell itself is not.

Segmentation and Cell Switching

Fetching a cell from memory differs from fetching a SAR User Payload from memory in both the size of the transfer (a cell is larger than a SAR-SDU) and the LAC and LO initialization values. Segmentation is usually triggered by an event such as a Peak Rate Pacing Counter timing-out. The APU switching a cell from an external memory-mapped device must be triggered by an external event.

Figure 8.4 shows the relationship between CS-PDU main-memory addresses and the VCR Cell Holder (Addresses for a standard 52-byte cell and a user specific 60-byte cell).

Figure 8.4
CS-PDU Main Memory and VCR Cell Holder Addresses



8.5 CRC32 Generation

CRC32s can be individually calculated for each CS-PDU actively undergoing either segmentation or reassembly. For CS-PDUs undergoing segmentation, the final CRC32 result is appended (under APU control) to Bytes [44:48] of the SAR-SDU of the last cell generated from the CS-PDU. For CS-PDUs undergoing reassembly, the CRC32 result is used as a checking mechanism by comparing it with the CRC32 received in the last cell of the CS-PDU. Because the ATMizer Architecture supports cell multiplexing and demultiplexing from up to 64K VCs, the APU must provide CRC32 Partial Result storage and retrieval services to allow for multiple concurrently active CRC32 calculations to be performed by the single CRC32 Generator.

As part of its Partial Results Management function, the APU must set the CRC32 Register to all ones prior to retrieving the first SAR-SDU for an AAL 5 CS-PDU. The 12-word DMA Read operation automatically generates a 32-bit CRC32 Partial Result in the CRC32 Register. The APU must retrieve this value at the end of the DMA operation and save it to preset the CRC32 Generator prior to the next transfer from the same CS-PDU. If more than one cell is to be built from a CS-PDU before proceeding to the next CS-PDU (the burst length is greater than one), and if no other DMA operation takes place in the interim, the APU need not retrieve and restore the CRC32 Partial Result until the final SAR-SDU has been retrieved from the CS-PDU. Before proceeding to the next CS-PDU, the AAL 5 CRC32 Partial Result must be stored in a place where it can be retrieved the next time that the CS-PDU is segmented (most likely in the CPE for the VC).

When the last SAR User Payload of a CS-PDU has been fetched from memory, the APU reads the CRC32 Final Result from the CRC32 Register and appends the result to the last four bytes of the cell in the VCR Cell Builder. If the final DMA transfer is set as 48 bytes, user software must ensure that the last four bytes of the CS-PDU in main memory (the CRC32 Field) are preset to all zeros. If the last transfer is executed as a 44-word transfer, no such restriction applies.

On reassembly, the APU must preset the CRC32 Register with all ones prior to initiating the first reassembly DMA operation for a CS-PDU. The APU again retrieves the CRC32 Partial Result at the end of the DMA operation, saving it away in the VCR or system memory (where ever CPEs are saved) and restoring it prior to reassembling the next cell of the CS-PDU. Again, if the last transfer is queued up as a 48-byte transfer, the APU must first set the CRC32 Field in the Cell Holder to all zeros before initiating the

DMA operation. At the end of the last transfer, the APU reads the CRC32 Final Result from the CRC32 Register and compares it to the result carried into the ATMizer Architecture in the last cell of the CS-PDU. If they differ, a CRC32 error has been detected and the ATMizer Architecture must inform the Host CPU utilizing the user-defined messaging system.

8.6 Misaligned Operations

The ATMizer Architecture DMAC can perform a DMA operation of any byte length less than or equal to 64 bytes, beginning at any VCR byte offset and any memory byte offset. In a Segmentation-Implementing Gather, for example, two physically disjunct data structures, one 53 bytes and one 91 bytes (87 bytes plus a blank 4-byte CRC32 Field) can form a single logical AAL 5 CS-PDU. The ATMizer Architecture must perform the following operations to segment this disjunct CS-PDU:

Note

The operations are based on these assumptions:

1. The CS-PDU Fragment 1 (53 bytes) starts at memory address 0x00000000.
2. The CS-PDU Fragment 2 (91 bytes) starts at memory address 0x00001000.
3. The next active Transmit Cell Builder in the VCR starts at 0x0100.

The operations are:

Step 1. Build the first cell with the DMA Control Register and Effective Address Fields (defined in Section 8.2, “Registers”) set as shown in Table 8.1.

Table 8.1 Field Settings for First Cell

Fields	***Setting***	***Function***
MAR, MAC, MOR	0x00000000	SAR-SDU Retrieval
BC	0x30 (48)	Transfer 48 Bytes
LAC	0x110	ATM Header is placed (starts) at 0x010C. Transmission Cell Builder starts at 0x0110.
LO	0x0 (00_2)	Offset starts at zero

Step 2. Build the first five bytes of the next cell with the DMA Control Register and Effective Address Fields set as shown in Table 8.2.

Table 8.2 Field Settings for the First Five Bytes of the Second Cell

Fields	*Setting*	*Function*
MAR, MAC, MOR	0x00000030	Get remainder of first fragment
BC	0x05 (5)	Transfer 5 Bytes
LAC	0x150	ATM Header starts at 0x014C. Transmission Cell Builder starts at 0x150
LO	0x0 (00_2)	Offset starts at zero

Step 3. Build the last 43 bytes of the next cell with the DMA Control Register and Effective Address Fields set as shown in Table 8.3.

Table 8.3 Field Settings for the Remaining 43 Bytes of the Second Cell

Fields	*Setting*	*Function*
MAR-MAC-MOR	0x00001000	Fill SAR-SDU from second fragment
BC	0x2B (43)	Transfer 43 Bytes
LAC	0x155	ATM Header starts at 0x014C. Transmission Cell Builder starts at 0x0155.
LO	0x0 (00_2)	Offset starts at zero

Step 4. Build the final cell with the DMA Control Register and Effective Address Fields set as shown in Table 8.4.

Table 8.4 Field Settings for Final Cell

Fields	*Setting*	*Function*
MAR-MAC-MOR	0x0000102B	Get remainder of second fragment
BC	0x30 (48)	Transfer 48 Bytes
LAC	0x190	ATM Header starts at 0x018C. Transmission Cell Builder starts at 0x0190.
LO	0x0 (00_2)	Offset starts at zero

The CRC32 Generator may be thrown off-track by the gap in the data stream used to build the cell when building AAL 5 transition cells. Building a cell from one or more word-aligned data structures does not greatly impact CRC32 generation when a data structure is always an even multiple of four bytes. User firmware simply retrieves the CRC32 Partial Result from the first DMA operation and restores it to the CRC32 Generator before the second DMA transfer starts and the CRC32 generation process proceeds without a problem.

If, however, the Gather function involves data structures that require non-word-aligned accesses, as shown in Step 2 above, the CRC32 generator is thrown out of alignment (because the CRC32 generator operates on 32 bits of data at one time). Therefore, firmware must first completely construct

the SAR-SDU in the VCR, using as many data structures as required to fill out the body of the cell and without regard to data structure alignment, before asking for a CRC32 calculation.

Once the SAR-SDU has been constructed in the VCR, the CRC32 Partial (or Final) Result is calculated by initiating a DMA Ghost Write operation to an arbitrary address. The DMA Ghost Write operation acts internally like a memory write operation. The DMAC performs a Ghost Write operation, internal to the ATMizer Architecture, at a rate of one word/cycle. Once the operation has completed, the CRC32 value can be read from the CRC32 Register the same as in any AAL 5 DMA Segmentation procedure.

Since the CRC32 Generator works on aligned data (data after it passes through the DMAC byte aligners), future cells built from the final CS-PDU fragment do not require Ghost operations. CRC32 generation proceeds smoothly as long as another unaligned boundary condition is not encountered.

On reassembly operations, if header stripping and data field alignment is employed for application acceleration, the same issues may arise with cells that contain the end of one header and the data field of a packet. On reassembly, the CRC32 Generator works on VCR data before it reaches the data aligners. Therefore, after the Ghost operation is done to generate the CRC32 for the transition cell, future operations to a single fragment need not utilize Ghost operations because the SAR-SDU is word aligned in the VCR, even though it may not be word aligned after being written into main memory. The CRC32 Generator uses data aligned to its VCR destination, not the main memory, in both directions.

All the incoming and outgoing cells are stored right-aligned in the VCR. For the DMA Controller to perform in the standard manner, the VCR/Main Memory Starting Address Offset and Byte Count must be one of the combinations shown in Table 8.5.

Table 8.5 Starting Address Offset and Byte Count

Address Offset	***Byte Count***
0x0	64
0x1	63
0x2	62
0x3	61
0x0	60
:	:

The DMA Engine in the DMAC always expects the last DMA operation to be a word read/store. If user firmware programs a 64-byte DMA operation starting with Address Offset 0x1, the ATMizer Architecture DMA first transfers three bytes (bytes 0x1, 0x2, 0x3), then groups of four bytes (bytes 0x4, 0x5, 0x6, 0x7 and so on until 0x40, 0x41, 0x42, 0x43), which ends up being a 67-byte DMA operation. The designer must be careful to build a system that does not write over useful information during a DMA write.

8.7 Scatter and Gather Operations

The ATMizer Architecture provides the system designer with all of the functionality needed to implement a fully robust scatter-gather ATM network-to-Host interface. In the Gather direction (during segmentation) the ATMizer Architecture is capable of generating cells from any number of separate data structures as if they were a single contiguous CS-PDU. This capability means that the Host Processor does not need to do a series of time consuming data movement operations to form a contiguous CS-PDU in a local buffer memory prior to initializing the Segmentation operation. When using the ATMizer Architecture in a TCP/IP application, the TCP/IP header may reside in a different location within Host memory from the actual user CS-PDU data payload. Also, the actual CS-PDU data payload field may actually consist of a number of discontinuous pages of memory. Because the ATMizer Architecture supports Gather operations, there is no need to move all of these data structures in advance into a single CS-PDU.

The actual implementation of Scatter and Gather functions are implemented in user firmware. In general, the Gather function can be implemented by having the Host Processor pass to the ATMizer Architecture a series of Segment CS-PDU Fragment messages with the appropriate user defined control structures. The APU, recognizing that it is involved in a Gather operation, is programmed not to generate End-of-CS-PDU Header fields at the end of a CS-PDU fragment. It is also programmed to resolve the arrival at an End-of-CS-PDU fragment boundary (automatically resolve the link list pointer or simply pass a message to the Host Processor asking it to resolve the next pointer for it).

8.8 DMA Operation Completion

The APU must determine that a DMA operation is complete before it attempts to use the information retrieved by the DMA operation. In the case of segmentation, the APU must determine that the DMA Controller has retrieved the entire SAR-SDU before it can queue the cell for transmission.

There are two methods for the APU to determine when a DMA operation is complete:

- Branch on Coprocessor Condition 3 True
- Interrupt

These methods are described in the following two subsections.

Note The CpCond and Interrupt signals are internal to the APU core which is part of the ATMizer Architecture. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information

---

Branch on Coprocessor Condition 3 True

The DMA Controller generates a DMA_Busy internal signal whenever it is involved in a DMA transfer. DMA_Busy is connected directly to the APU's CpCond3 input pin. Programmers familiar with the R3000 CPU architecture understand that the four CpCond inputs to the R3000 can be tested using a conditional branch instruction. If the APU wishes to determine if the DMAC is busy, it can execute a Branch on Coprocessor Condition 3 True instruction three clock cycles after the DMA Controller is programmed. If CpCond3 is True (DMA_Busy is asserted), the DMA Controller is still busy and the APU should not attempt to use the data (queue the cell for transmission). If CpCond3 is False (DMA_Busy is not asserted) the DMA Controller has finished its operation and the data is valid in the VCR. The APU is free to queue the cell for transmission or read the retrieved data from the VCR.

If the APU attempts to program a DMA operation into the DMA Controller before the DMA Controller has completed a pending operation, the APU stalls until the DMA operation is completed. As soon as the existing operation completes, the new operation is loaded into the DMAC and the APU continues.

Interrupt

Internally, there is a signal connected to APU Interrupt3. When the byte count during DMA transfer reached its terminal count, the DMA interrupt signal is asserted to notify the APU that DMA operation is complete. This interrupt is maskable by clearing the corresponding bit in the Hardware Interrupt Mask Field of the APU Status Register (see Chapter 14). The interrupt is cleared by writing to the DMAC Control Register to start another DMA operation. The interrupt routine should check if there are any more DMA operations to be done and if there are, the code should

program the DMAC Control Register to start another DMA transfer. If there are no DMA operations pending, the interrupt routine should mask this interrupt.

Chapter 9
ATM Cell Interface (ACI)

This chapter describes the function and operation of the ATM Cell Interface.

This chapter has eight sections:

9.1 Overview

The ATM Cell Interface (ACI) is the ATMizer Architecture’s eight-bit interface to the ATM port-side circuitry. The ACI contains both the ATM port-side transmitter and receiver functions and connects directly to the Transmission Convergence Sublayer (TCS) framing circuitry. The ACI Receiver logic receives data from the external framing logic and reconstructs ATM cells in the VCR. The ACI Transmitter logic transfers cells from the VCR to the external framing logic.

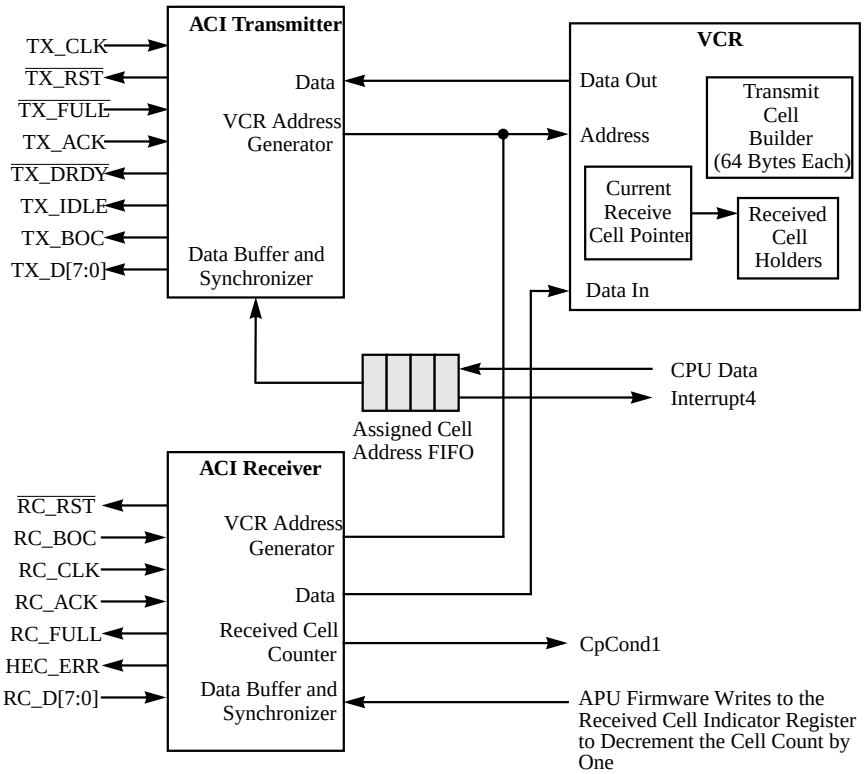
The ACI contains data buffers and frequency decoupling logic so that the ATMizer Architecture ATM ports can be directly connected to the ATM line transceivers. All metastability issues are addressed and solved by the ATMizer Architecture. The ACI protocol conforms to the UTOPIA PHY-to-ATM Layer Specification (Version 1.22) and the SAI Specification (Version 2.3).

The ACI Transmitter takes cells that have been built in the VCR and transfers them one byte at a time to an external ATM line. The ACI Transmitter can be programmed to generate and insert the HEC and generate and append a CRC10 Field to AAL 3/4 cells. The ACI Transmitter also decouples Cell Rates. If there is not an assigned cell in the VCR ready for transmission, the ACI Transmitter automatically sends an Idle Cell.

The ACI Receiver accepts cells, one byte at a time, from the ATM line and reconstructs these cells in the VCR so that the APU may process them (either reassemble the cell or switch the cell).

Figure 9.1 shows a block diagram illustrating how the ACI Transmitter and Receiver are connected to other logic in the ATMizer Architecture.

Figure 9.1 ACI Transmitter and Receiver Block Diagram



9.2 ATM Cell Size

The user can program the size of an ATM cell (up to 64 bytes) to support applications that employ extra header fields to convey switch specific information. The typical ATM cell in the VCR is 52 bytes, but cells can be 56, 60, or 64 bytes. The HEC value is generated and inserted into the cell as it is passed out of the ATMizer Architecture. Therefore, the actual ATM cell on the line could be 53, 57, 61, or 65 bytes.

The APU firmware creates cells from memory-mapped CS-PDUs, Real-time Data Streams, or from existing memory resident cells. The cells are built in cell holding areas inside the VCR. Once built, the APU firmware transfers the cells, one byte at a time, through the ACI to the TCS Framing Circuitry. The ACI contains special buffering circuitry to decouple the ATMizer Architecture System Clock frequency from the clock frequency required by the Transmission Convergence Sublayer framing circuitry. The ACI is driven by the ATM line-derived byte clocks.

In ATM, raw cell data is combined with certain overhead information to form transmission frames. The logic that performs this framing is in the Transmission Convergence Sublayer. ATM supports framing modes that insert several framing bytes into each transmission frame. As a result, bytes are received that do not correspond to data transfers between the TCS Framing Logic and the ACI Ports. The system may need to gap data transfers to and from the ACI Ports, so there must be a way to signal to the ATMizer Architecture when no data transactions are desired (to create gaps in the data stream). In the ATMizer Architecture application, external logic can indicate that a data transfer is not desired by either stopping the ACI Port clock(s) (running the ACI Ports off of gapped clocks) or deasserting TX_ACK or RC_ACK (running the ACIs off of the free running line clocks and using a data acknowledge mechanism to deal with gapping).

9.3 Frequency Decoupling

The ATMizer Architecture contains all of the logic necessary for decoupling the ATMizer Architecture's internal clock (the System Clock) from the clock rates of the transmission lines. The system designer clocks byte-wide data out of the ATMizer Architecture that is used to drive the transmission line and clocks data into the ATMizer Architecture derived from the received data stream. All frequency decoupling and metastability issues are dealt with inside the ACI circuitry. The ATMizer Architecture uses a simple handshake acknowledgment mechanism to allow external logic to pause data transfers between the ATMizer Architecture and the line transceivers. The pause may be required if external logic suspends the

cell stream in order to generate and send or extract Transmission Convergence Sublayer framing overhead.

9.4 ACI Transmitter

The ACI Transmitter transfers cells from the VCR to the ATM Transmission Convergence Sublayer framing logic.

Transmitter Cell Sources

There are three possible ways for cells to become available for transmission in the VCR:

- Segmentation
- Internal Switching
- External Switching

Segmentation

In response to an internal or external event, the APU determines that it must segment one or more CS-PDUs or generate a cell from one or more realtime data buffers. The APU chooses an available Transmit Cell Holder to be used in the cell building process. In order to accomplish segmentation, the APU initiates a DMA Read Operation to transfer the SAR-SDU from a memory based CS-PDU or Realtime Data Buffer into the VCR. The APU provides the DMA Controller with all of the proper address information such that the SAR-SDU is transferred into the Transmit Cell Holder in its proper cell location. The APU then generates or retrieves and appends the necessary headers and trailers to the cell and queues the cell for transmission. The APU queues the cell for transmission by writing the VCR Starting Address of the cell into the Transmitter Cell Address FIFO.

Internal Switching

The ATMizer Architecture is capable of transferring cells that arrive over the ACI Receiver (RC_D[7:0]) out of the ATMizer Architecture utilizing the ACI Transmitter without having to pass the cell to main memory. This process works as follows:

1. All cells arriving into the ATMizer Architecture over the ACI Receiver Port are written into the VCR.

2. The ATMizer Architecture sets aside the first 256 bytes (4 Cells), 512 bytes (8 Cells), 1024 bytes (16 Cells), or 2048 bytes (32 Cells) of VCR memory for Received Cell Buffering.
3. Once a cell is written into the VCR the APU must process the cell. As with all operations, the APU uses cell header fields as an index into a VCR or memory based lookup table that contains information on how the cell should be processed.
4. If the lookup yields information that indicates that the cell should be sent out over the ACI Transmitter, the APU can perform any necessary header manipulation operations (such as VCI or VPI translation and/or congestion notification insertion) before queuing the cell for transmission.
5. The APU queues the cell for transmission by writing the VCR starting address of the cell into the Cell Address FIFO.

External Switching

In certain applications, the ATMizer Architecture has access to main memory-based cells that have arrived over some other ATM port, but need to be transferred out over the ATMizer Architecture ACI Transmitter. Some user defined external event mechanism (through assertion of GPINT_AUTO or through APU polling of some mailbox location) informs the ATMizer Architecture of the need to switch a main memory resident cell. If the ATMizer Architecture finds that a cell exists externally (the location of which is likely to be known by convention), it can initiate a DMA operation to bring the cell into the ATMizer Architecture. Once inside, the cell headers can be modified by the APU (or they may have already been modified by the ATMizer Architecture that placed the cell in external memory). Once the cell has been fully retrieved from memory and placed in the VCR, the APU queues the cell for transmission by writing the VCR starting address of the cell into the Cell Address FIFO.

Queuing a Cell for Transmission

Transmission Cells can be generated in one of three fashions. What is common to each of the scenarios listed previously is that the APU queues a cell for transmission by writing an address pointer into the Cell Address FIFO. This address pointer points to where the cell begins in the VCR. The address is passed through the use of a Store Word Instruction with Effective Address Bits [31:0] = 0xFFF04500 if CRC10 does not need to be gen-

erated and EA[31:0] = 0xFFF04540 if CRC10 is to be generated for the cell to be transmitted.

If the APU attempts to write an address to the Cell Address FIFO, but the Cell Address FIFO is already full, the write operation will cause the APU to stall. The APU remains in the stall operation until the ACI Transmitter finishes sending a cell and a location becomes available in the Cell Address FIFO. The status of the FIFO can be checked by reading the System Control Register Bits [2:0], which indicates how many addresses are left in the FIFO. The APU can prevent writing an address into a full buffer (and prevent the delays associated with it) by testing the state of the buffer before beginning a segmentation or cell switching application.

Another way to prevent the APU from stalling due to a full address FIFO is to enable APU Interrupt4. Interrupt4 is very useful for firmware coding. The Assigned Cell Address FIFO can store up to four addresses. When there is only one address left in the Cell Address FIFO, the ACI asserts Interrupt4. Then the interrupt routine can fill the FIFO all at once by writing three more transmission cell addresses. The write operation clears Interrupt4.

Note The CpCond and Interrupt signals are internal to the APU core which is part of the ATMizer Architecture. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

Cell Rate Decoupling

The Cell Address FIFO mentioned above is a four-word FIFO that holds the VCR addresses of cells that are ready for transmission. When the ACI Transmitter reaches the end of a cell, it checks the Cell Address FIFO to see if an address exists for a completed cell. If it does, the ACI Transmitter automatically begins fetching the new cell from the VCR and sending it, one byte at a time, to the external transmission convergence framing logic over TX_D[7:0]. If an address does not exist in the Cell Address FIFO when the end of the present cell is reached, the ACI Transmitter performs Cell Rate Decoupling.

As part of its start-up code and prior to initiating transmitter operations, the APU must build a complete Idle Cell in the last 64 bytes of the VCR space. The Idle Cell pattern should be the same length as the user defined Transmit Cell Size. By designating an area in the VCR as the Idle Cell Holder, a user is free to generate an Idle Cell that matches his switch specific structure.

During normal operation, if the ATMizer Architecture reaches the end the current cell and no other address is available in the Cell Address FIFO, it sends the Idle Cell that resides in the last 64 bytes of VCR location. The ATMizer Architecture asserts TX_IDLE to inform external logic that the cell being transmitted is an Idle Cell. Please refer to the timing waveforms in Chapter 13 for detailed timing of TX_IDLE assertion and deassertion.

Preparation for Transmission

When the ATMizer Architecture powers up, the VCR content is undefined. As part of its reset routine, the APU must create the Idle Cell pattern in the VCR, select the HEC Transmit Mode in the System Control Register, and select the Transmit Cell Size in the System Control Register.

After the Transmit Offset Field in the System Control Register is completely defined and the Idle Cell Pattern has been built, the program must perform a second write to the System Control Register to set the Transmit Initialize Bit. The first cell that the ATMizer Architecture sends out has to be an Idle Cell, then the APU queues an assigned cell for transmission by writing its start address into the Cell Address FIFO. Then the ACI Transmitter sends the assigned cell after reaching the end of the current Idle Cell transmission.

Firmware synchronizes the ACI Transmitter by:

1. Initializing the ACI Transmitter
   - setting the Transmit Cell Size by writing a value to the Transmit Offset Field in the System Control Register
   - building an Idle Cell in the VCR
   - configuring the HEC Transmit Mode
2. Setting the Transmitter Initialize Bit in the System Control Register

   Setting this bit puts the ACI Transmitter in the normal operation. The ACI deasserts $\overline{\text{TX\_RST}}$. The ACI Transmitter transmits Idle Cells until the APU firmware writes the first Assigned Cell address in the Cell Address FIFO. The first time firmware sets the Transmitter Initialize Bit, the ACI Transmitter asserts $\overline{\text{TX\_DRDY}}$ to indicate that it is ready to transmit.

The ATMizer Architecture indicates that it has retrieved the first byte of data by asserting its $\overline{\text{TX\_DRDY}}$ and TX_BOC outputs. After system reset or transmitter synchronization, external logic must wait for the ATMizer

Architecture to assert $\overline{\text{TX\_DRDY}}$ and TX_BOC before asserting TX_ACK. Once $\overline{\text{TX\_DRDY}}$ is asserted it will remain asserted and data will continue to be sourced onto TX_D[7:0] as long as TX_CLK remains within specification and $\overline{\text{TX\_FULL}}$ is not asserted.

When the PHY Layer asserts $\overline{\text{TX\_FULL}}$, the Transmitter deasserts $\overline{\text{TX\_DRDY}}$ ($\overline{\text{TX\_DRDY}}$ can be connected to the PHY Layer TX_ENABLE signal), and external logic should deassert TX_ACK. For more information, please refer to Section 9.9, "Interfaces."

1. CRC10 During Transmission

   Setting Effective Address Bit 6 (during the store to the assigned Cell Address FIFO) informs the Transmitter that it must generate and insert a CRC10 value for the next cell to be transmitted. Therefore, the Effective Address of the assigned Cell Address FIFO for cells that need CRC10 is 0xFFF04540, and the Effective Address for cells without CRC10 is 0xFFF04500.

2. HEC During Transmission

   Setting first bit of the HH Field in the System Control Register (Bit 21) to one, causes the ACI Transmitter to enable HEC generation on transmission. HEC is placed after the cell header. So, for a 52-byte cell, the ACI Transmitter transmits 53 bytes and for a 64-byte cell, the ACI Transmitter transmits 65 bytes.

9.5 ACI Receiver

The ACI Receiver accepts bytes of cell data from the ACI Receiver Data Bus, RC_D[7:0], and uses these bytes of data to reconstruct cells in the VCR. The ACI Receiver also informs the APU that a cell has arrived by asserting CpCond1. Upon detecting the arrival of a cell, the APU can read the cell header and use it as an index into a VCR-based or memory-based lookup table. From this lookup, the APU determines the AAL type (layer number) used for the VC and the operations that must be performed on the cell.

Received Cell Handling Options

The Received Cells can be handled in one of four ways:

- Reassembly
- Internal Switching
- External Switching

- Discarding

Reassembly

The APU can choose to reassemble the cell into a CS-PDU in memory by initiating the appropriate DMA operations. In the case of reassembly, the DMA Controller is configured with the VCR address of the SAR-SDU, the memory address of the CS-PDU and the appropriate transfer length count. The DMA Controller then automatically accomplishes the reassembly operation through a series of memory write transfers.

Internal Switching

The ATMizer Architecture can use the ACI Transmitter to transfer cells, received by the ACI Receiver, out of the ATMizer Architecture, without ever passing the cell out to main memory.

External Switching

In certain applications, the ATMizer Architecture needs to pass entire cells, headers and trailers intact, to some other ATM port interface that has access to the same memory space as the ATMizer Architecture (perhaps another ATMizer Architecture). In such a situation, the ATMizer Architecture may choose to first execute one or more header manipulation operations before transferring the cell to the centralized memory structure. After performing these operations, the ATMizer Architecture initiates a DMA operation to transfer the cell to memory so that another ATM port interface can gain access to it. After transferring the cell to memory the ATMizer Architecture can alert another port interface to the availability of the cell by writing to a memory mapped mailbox location.

Discarding

The APU firmware can discard the cell by writing to the Received Cell Indicator Register without initiating any DMA operations if the APU firmware detects a CRC10 error. The write operation makes the Current Received Cell Address Register point to the next received cell in the VCR.

Received Cell Indication

This subsection explains how the APU recognizes that cells are waiting to be processed in the VCR.

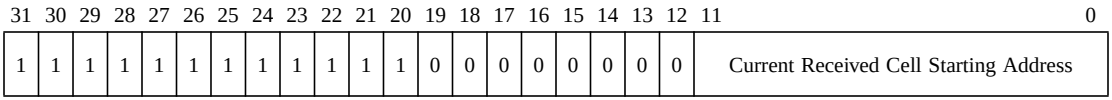
The APU firmware can check for the presence of Received Cells that have yet to be processed by periodically polling CpCond1 using the Branch on CpCond1 True instruction. If the APU firmware detects that CpCond1 is asserted, a cell is available, and it can begin processing the cell.

The APU finds out the location of a received cell in the VCR by reading the Current Received Cell Address Register (Effective Address 0xFFF04400), which contains the starting address of the cell waiting to be processed next. The logic in the ACI Receiver that generates the CpCond1 signal is simply an up/down counter. Each time a cell arrives, the counter counts up by one.

Each time the APU has processed a cell, APU firmware lowers the counter by writing to the Received Cell Indicator Register. (The Received Cell Indicator Register is not really a register. It is an address decode circuit with an Effective Address of 0xFFF0460C.) At the same time, the pointer in the Current Received Cell Address Register is incremented by 0x64, which points to the next cell if there is one.

Figure 9.2 shows the Current Received Cell Address Register. The Current Received Cell Address Register is a pointer that points to the Received Cell Holder that currently needs servicing by the APU. After the APU firmware writes to the Received Cell Indicator Counter at the end of the Received Cell routine, the ACI changes the Current Received Cell Address Register to point to the next Received Cell to be serviced.

Figure 9.2
Current Received Cell Address Register



The Current Received Cell Address Register's Effective Address is 0xFFF04400.

The Received Cell Indicator Register controls the Received Cell Counter in the ACI. Each time a cell arrives, the ACI increments the Received Cell Counter. While the Received Cell Counter is greater than zero, the ACI asserts CpCond1. When the APU has finished servicing a cell, and written to the Received Cell Indicator Register, the ACI decrements the Received

Cell Counter by one. After all cells are serviced, the ACI clears the Received Cell Counter to zero and deasserts CpCond1.

If the APU becomes occupied handling certain boundary conditions or gets blocked from the memory backplane for a period of time, cells will begin piling up in the VCR and the Received Cell Count will continue to rise. Once the APU frees up, it should immediately begin draining the Received Cell Buffer. Each time it processes a cell it reduces the Received Cell Counter by one and then immediately checks to see if additional cells are in the VCR by polling the CpCond1 input again. If CpCond1 remains asserted, cells have accumulated in the ACI Receiver and should be drained before processing any pending segmentation requests.

User firmware may use Interrupt5 (the Received Cell buffer is half full) to handle the Received Cell operation. Upon interrupt, the user firmware should jump to the Received Cell subroutine and start draining those cells.

The system designer may wish to interleave segmentation handling in with Received Cell draining. This can be done, but it prolongs the period of time required to drain the Received Cell Buffer and increases the chance that a busy backplane will cause eventual Received Cell Loss.

The APU can check the number of cells in VCR that need to be serviced by reading the System Control Register. System Control Register Bits [13:8] (Effective Address 0xFFF04A00) contain the number of cells in the Received Cell Holder that need processing. When Bits [13:8] are 000000_2, there are no cells. When Bits [13:8] are 100000_2, there are 32 cells in the Received Cell Holder that need to be processed.

If the Received Cell Buffer overflows, the ACI Receiver does not write cells into the VCR until a location is available. The ACI Receiver asserts the overflow signal output, RC_FULL, to indicate an overflow.

There are two ways for the ACI Receiver to handle an overflow:

1. If the BM Bit in the System Control Register is cleared to zero, the ATMizer Architecture expects to handshake with a UTOPIA device. The ATMizer Architecture will take only one more byte when it asserts RC_FULL.
2. If the BM Bit in the System Control Register is set to one, the ATMizer Architecture expects to handshake with an SAI device. The ATMizer

Architecture will finish receiving the rest of the bytes from the current cell and then stop.

Receiver Reset

The ACI Receiver is reset by firmware. Upon a power-on reset, and after $\overline{\text{RST}}$ is deasserted, the ATMizer Architecture does not deassert $\overline{\text{RC\_RST}}$. Firmware must initialize all of the ACI Receiver parameters, such as Cell Holder Size, Cell Size, and HEC handling, before setting the Receiver Initialize Bit in the System Control Register to one. Setting this bit to one deasserts $\overline{\text{RC\_RST}}$. $\overline{\text{RC\_RST}}$ should be connected to the Physical Layer Reset. When $\overline{\text{RC\_RST}}$ is deasserted, the ACI Receiver is then ready to receive cells. If, before the Receive Initialize Bit is set to one, the ACI Receiver detects the assertion of RC_BOC, it sets the RC_BOC Bit (Bit 6 of the System Control Register) to notify the APU.

Firmware must perform two stores to the System Control Register to initialize the ACI Receiver. The first store defines the ACI Receiver parameters and the second sets the Receive Initialize Bit to one.

9.6 Traffic Shaping

User firmware can use the Global Pacing Rate Register (GPRR) for traffic shaping (controlling the transmission rate). When the network experiences congestion, the GPRR provides a fast way to slow down the transmission rate of the ACI. A single APU instruction modifies the GPRR, which determines the percentage of assigned cells sent out over the ATMizer Architecture ACI Transmitter (transmission port). The amount of initial data reduction, as well as the algorithm by which the ATMizer Architecture returns to full-speed operation, can be implemented in APU firmware. Algorithms can be modified as more is learned about ATM network congestion.

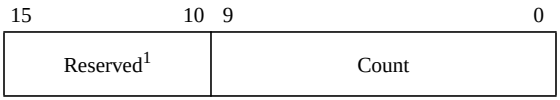
Average and Peak Rate Pacing and Burst Length are useful for managing the bandwidth used by a particular VC. OAM software can manipulate these values for active VCs to manage the overall data throughput rate (or information rate) on the Transmission line. However, it is almost impossible to effectively shape the overall ATM port information rate through this mechanism. Shaping the overall information rate may be necessary when connecting into a system that can only handle a limited information rate or during periods of high congestion in a switching network. In the case of a congested network, the latency between congestion notification and the Host Processor's ability to modify the pacing parameters may be high. As a result, many cells are sent into a congested network and are lost, requiring the retransmission of many CS-PDUs. This further exacerbates the

congestion problem. And by the time the system responds to the notification of congestion, the congestion situation in the network may have actually changed.

The ATMizer Architecture implements a Global Pacing Rate Unit, which contains a 10-bit countdown counter and a register (GPRR). The GPRR contains a value that is the maximum number of continuous assigned cells. This Global Pacing Rate control mechanism is a quick way to limit the overall transmission bandwidth used on the Transmission Port (ACI Transmitter). When the counter reaches zero, the ACI Transmitter forces an Idle Cell onto the outgoing cell stream.

Figure 9.3 shows the Global Pacing Rate Register. The Store Word instruction should be used to write to this register.

Figure 9.3 Global Pacing Rate Register



1. All reserved bits must be 0.

The Global Pacing Rate Register Effective Address is 0xFFF047X0.

If EA7 = 0, the GPRR function is disabled and no Idle Cell is transmitted. If EA7 = 1, the GPRR is enabled and an Idle Cell is inserted when the GPRR Counter counts down to zero.

Figure 9.4 shows the Maximum Line Utilization Rate when Count = 1 (50% Assigned Cells). Figure 9.5 shows the Maximum Line Utilization Rate when Count = 2 (67% Assigned Cells).

Figure 9.4 Maximum Line Utilization Rate (Count = 1, 50% Assigned Cells)

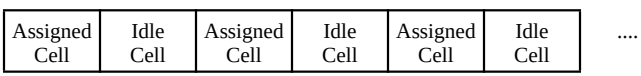
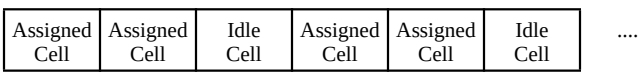


Figure 9.5 Maximum Line Utilization Rate (Count = 2, 67% Assigned Cells)



Note Even when the GPRR is disabled, Idle Cells are transmitted if there is no active cell.

9.7 HEC Generation and Checking

In applications that generate and check their own HEC values, HEC generation can be disabled by clearing the HH0 Bit in the ATMizer Architecture System Control Register. The ATMizer Architecture supports a complete decoupling between HEC handling on transmitting and on receiving. The ATMizer Architecture also allows for HEC errors to be ignored on receives while still expecting that an HEC value will be passed to the ATMizer Architecture from the Physical Layer. In some systems, the Transmission Convergence Sublayer performs error detection and corrects bit errors on the ATM Header but does not generate a new HEC. In this case, the ATM header is corrected but the HEC value is still incorrect. The ATMizer Architecture is equipped to handle such situations.

Some systems supporting external HEC generation require that a slot be present in the transmit data stream in the Byte 5 (fifth byte) of the cell for inserting the externally generated HEC. These systems should enable HEC generation and just overwrite the fifth byte with the externally generated HEC value.

The ATMizer Architecture supports the following HEC modes:

1. HEC generated on transmit and placed in Byte 5 of the cell
2. HEC not generated on transmit
3. HEC expected on Byte 5 and checked on receive
4. HEC expected on Byte 5 but not checked on receive (this is for the SONET system where framing logic corrects the header, but leaves the HEC Field incorrect)
5. HEC not expected in Byte 5 and therefore not checked on receive

Section 14.1, “System Control Register” shows how the HH Field in the System Control Register defines HEC generation modes.

9.8 CRC10 Generation and Error Checking

Since the ATMizer Architecture can mix cells of all AAL types within a single ATM Cell stream, the APU must indicate to the ACI Transmitter when an AAL 3/4 cell is to be sent. For AAL 3/4 cells, the ACI Transmitter calculates a CRC10 value for the SAR-PDU and appends it to the cell. The APU indicates to the ACI Transmitter that CRC10 generation is required by setting APU Address Bit 6 to one for the Store operation that writes the address of the cell into the ACI Transmitter's Cell Address FIFO. If the APU clears Address Bit 6 to zero, the ACI Transmitter transmits the value contained in the last two bytes of the Transmit Cell in the VCR 64-byte Cell Holder location for the addressed cell.

For AAL 3/4 cells, the APU must construct the ATM Header and the SAR Header from the cell and place them into the appropriate locations in the VCR (cells are always bottom justified within a 64-byte Cell Holder area in the VCR). The APU must also calculate the length of the actual SAR-SDU in bytes and place it into the Length Indication (LI) Field of the AAL 3/4 trailer before the cell can be sent through the Transmitter. The ACI Transmitter transmits the LI Field out of the ATMizer Architecture in Bits [7:2] of Byte 47 of the SAR-PDU. The ACI Transmitter inserts the CRC10 value that it calculates into Bits [1:0] of Byte 47 and Bits [7:0] of Byte 48 of the SAR-PDU. (Note that Byte 47 of the SAR-PDU is located at Byte 62 of the Cell Holder and Byte 48 of the SAR-PDU is located at Byte 63 of the Cell Holder, the last byte of the 64-byte Cell Holder.) The APU writes the LI Field into Byte 62 right justified. The Transmitter left justifies the LI Field before merging it with the two most significant bits of the CRC10 and then transmits it off chip. The Transmitter is designed to receive the LI Field right justified in Byte 62 of the Cell Holder (Byte 47 of the SAR-PDU) so that the APU does not have to perform a shift left instruction to left justify the value.

Note 1

If the SAR-PDU is less than 44 bytes, the APU must fill all unused locations in the Cell Holder with zeros. If no pattern is explicitly written into the unused bytes, the previous contents (whatever is stored in these locations in the VCR) is transmitted and is included in the CRC10 calculation. If it is important to zero out all unused fields, the APU should use the DMA Controller to stream zeros in from a reserved 64-byte location in memory containing nothing but zeros. This allows the APU to perform other tasks while the Cell Holder is being filled by zeros. Alternatively, the APU can write zeros to all unused bits.

Note 2

The DMA Controller does not clip VCR or memory writes at the end of the write. For instance, when the DMA Controller writes the last two bytes of the AAL 3/4 SAR-SDU into Bytes 60 and 61 of the Cell Holder in the VCR, it also writes Bytes 62 and 63. Therefore, in most cases, software should not write the LI Field value into Byte 62 of the Cell Holder until the DMA operation has completed.

The ATMizer Architecture can intermix cells of all AAL types (numbers) in a single cell stream. The ATMizer Architecture can also store up to 32 cells in the VCR. Since the ACI Receiver does not know the AAL type when a cell arrives, it should not just check for CRC10 and discard the cell if it has an error. The CRC10 circuitry calculates CRC10 on all incoming cells and stores the result in an internal 32-bit register. In the cell reassembly routine, the code detects the AAL type after getting the CPE for the cell. If the cell is AAL 5, then CRC10 is ignored. But, if the cell is AAL 3/4, for example, the code should check the CRC10 validity. This checking can be done in two instructions. The first instruction is a Load Halfword that loads Bits [15:0] of the System Control Register (Bit 15 is the CRC Error Bit) into a target register. If the CRC Error Bit is set to one, it indicates that there is a CRC10 error in the cell. The Load Halfword instruction extends the sign of Bit 15 in the target register. The second instruction is a Branch On Less Than Zero (applied to the target register). If there is a CRC10 error, this instruction causes a branch to an error handling routine.

9.9 Interfaces

The ATMizer Architecture ACI can be programmed to work with either UTOPIA or SAI devices. When the BM Bit (Bit 24) of System Control Register is cleared to zero (see Section 14.1, “System Control Register”), the ATMizer Architecture expects to communicate with UTOPIA interface on the ACI side. When the BM Bit is set to one, the ACI communicates with SAI devices.

UTOPIA

The proposed standard interface between the Physical (PHY) Layer and the ATM Layer is a subset of the ACI Interface. The Universal Test & Operations PHY Interface for ATM (UTOPIA) is a proposed standard to interface the multiple different PHY layers to the ATM Layer. There are currently (July 1994) four PHY layers defined by the ATM forum and a fifth one under study. By implementing the UTOPIA interface as a subset of the ACI, the user can choose from a variety of PHY layers for their system requirement. In the following explanation, the PHY Layer means the

transmission convergence logic external to the ATMizer Architecture and the ATM Layer means the ACI within the ATMizer Architecture.

Transmit

On the transmit side, the following signals are defined by the proposed standard, and they can be directly connected to the ACI Transmitter. Note that input and output refers to direction in relation to the PHY Layer.

TxData[7:0] **Transmitter Data** **Input**
These signals are byte-wide data driven from the ATM Layer to the PHY Layer. These signals are connected to the ACI's TX_D[7:0] signals.

TxSOC **Transmitter Start of Cell** **Input**
The ATM Layer asserts this signal HIGH when TxData[7:0] contains the first byte of the cell. This signal is connected directly to the ACI's TX_BOC signal.

$\overline{\text{TxEnb}}$ **Transmitter Enable** **Input**
The ATM Layer asserts this signal LOW during cycles when TxData[7:0] contains valid cell data. $\overline{\text{TxEnb}}$ should be connected directly to the ACI's $\overline{\text{TX\_DRDY}}$ signal.

$\overline{\text{TxFull}}$ **Transmitter Full** **Output**
The PHY Layer asserts this signal LOW at least four cycles before the PHY Layer is no longer able to accept transmit data. This signal is connected directly to the ACI's $\overline{\text{TX\_FULL}}$ signal. When the PHY Layer asserts $\overline{\text{TxFull}}$, the ACI deasserts $\overline{\text{TX\_DRDY}}$ which is connected to the PHY Layer $\overline{\text{TxEnb}}$ on the next clock cycle. When $\overline{\text{TX\_DRDY}}$ is deasserted, the ACI does not source data onto the TX_D[7:0] lines.

TxClk **Transmitter Clock** **Input**
This is a data transfer/synchronization clock input to the PHY Layer. The ACI in the ATMizer Architecture is design to accept any clock input up to 25 MHz. The clock supplied to the ACI TX_CLK should also be connected to TxClk on the PHY Layer.

Receive

On the receive side, the following signals are defined by the proposed standard and they can be directly connected to the ACI Receiver. Note that input and output refers to direction in relation to the PHY Layer.

RxData[7:0] **Receiver Data** **Output**
These signals are byte-wide data driven by the PHY Layer to the ATM Layer. These signals are connected directly to the ACI's RC_D[7:0] signals.

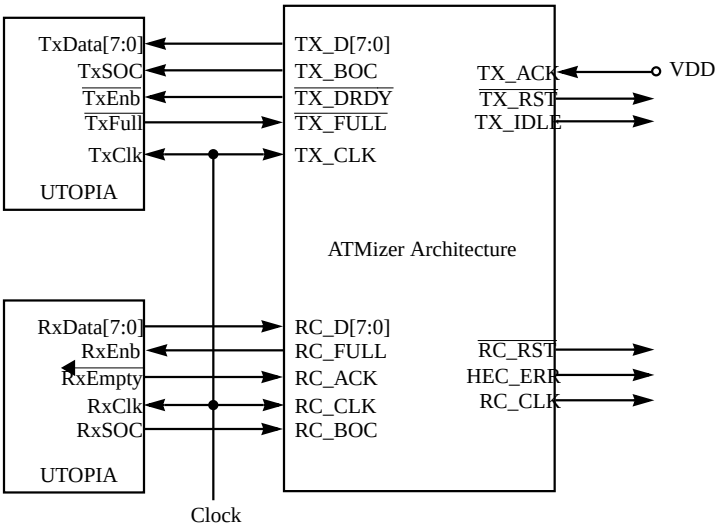
RxSOC **Receiver Start of Cell** **Output**
The PHY Layer asserts this signal HIGH when RxData[7:0] contains the first byte of a cell. This signal should be connected directly to the ACI's RC_BOC signal.

$\overline{\text{RxEnb}}$ **Receiver Enable** **Input**
The ATM Layer asserts this signal LOW to inform the PHY Layer to sample RxData[7:0] on the next rising edge of the clock. This signal should be directly connected to ACI's RC_FULL signal. When the receive buffer in the VCR is full, the ACI asserts RC_FULL which disables $\overline{\text{RxEnb}}$.

$\overline{\text{RxEmpty}}$ **Receiver Empty** **Output**
The PHY Layer asserts this signal LOW to inform the ATM Layer that the PHY Layer buffer is empty and there is nothing to give to the ATM Layer. When $\overline{\text{RxEmpty}}$ is asserted, there is no valid data on the current cycle. This signal can be directly connected to the ACI's RC_ACK signal. When the PHY Layer asserts $\overline{\text{RxEmpty}}$ LOW, the data in RxData[7:0] is invalid, and since $\overline{\text{RxEmpty}}$ is connected to RC_ACK, the ACI ignores the data in RC_D[7:0].

Figure 9.6 shows the UTOPIA Connection to the ATMizer Architecture.

Figure 9.6 UTOPIA Connection to ATMizer Architecture



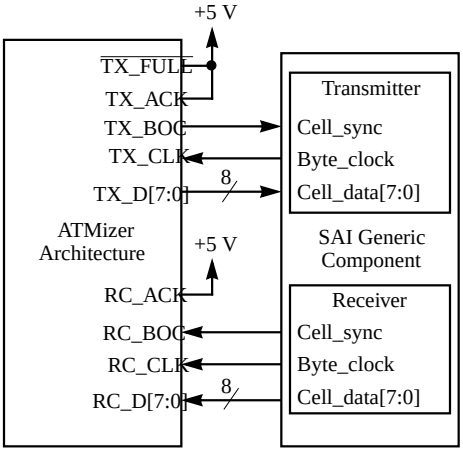
SAI

The Standard ATM Interface (SAI) is a European ATM-cell-based standard in which both the transmitter and receiver use only three signal categories to transmit and receive (Cell_sync, Byte_clock, Cell_data[7:0]). Cell_sync is Transmit and Receive Beginning of Cell signal, Byte_clock is Transmit and Receive Byte Clock signal, and Cell_data[7:0] is Transmit and Receive Byte Data Bus. Since the SAI functions like a subset of the UTOPIA Interface, this section only explains the differences between the SAI and UTOPIA modes.

For both transmit and receive, an SAI device aligns cell boundaries using TX_BOC and RC_BOC. An SAI device sources or latches data on every byte clock until it reaches 53 bytes. Then, another TX_BOC or RC_BOC may be asserted to align cells perfectly. If not, an SAI device will wait for the next TX_BOC or RC_BOC and then start counting another 53 bytes.

Figure 9.7 illustrates the connection between the ATMizer Architecture and a generic SAI device.

Figure 9.7 ATMizer Architecture-to- SAI Device Connections



In SAI Mode, the ATMizer Architecture still asserts TX_IDLE, HEC_ERR, and TX_DRDY signals as in UTOPIA Mode, but the SAI device is not connected to them.

In UTOPIA Mode during transmission, the ATMizer Architecture asserts TX_BOC for Idle Cells. In SAI Mode, during transmission, the ATMizer Architecture does not assert TX_BOC for Idle Cells.

In UTOPIA Mode during reception, after the ATMizer Architecture asserts RC_FULL, it receives one more byte and then stops. In SAI Mode during reception, after the ATMizer Architecture asserts RC_FULL, it receives the remaining six bytes of the cell and then stops.

The SAI Transmitter does not distinguish between Assigned Cells and Idle Cells. By setting the BM Bit to one, the ACI does not assert TX_BOC for Idle Cells and it makes the SAI device take all the Assigned Cells and ignore all the Idle Cells. On the receive side, UTOPIA states when RC_FULL is asserted, only one more byte should be taken. Therefore when BM = 0, the ATMizer Architecture asserts RC_FULL, takes one more byte, and then stops. For SAI mode, it assumes its buffer will never get full, therefore, when RC_FULL is asserted and BM is set to one, the ATMizer Architecture will take the remaining six bytes of the current cell and then stop. In this case, the external SAI device will keep sending cells (asserting RC_BOC) to the ATMizer Architecture. If the VCR is not free yet, the ATMizer Architecture will set the RC_BOC Error Bit in the System Control Register.

Chapter 10
Secondary Port (SP)

This chapter describes the function and operation of the Secondary Port.

This chapter has four sections:

10.1 Overview

The 32-bit Secondary Port (SP) is a 32-bit multiplex address, data, and control path. It can be used to perform a variety of data transfers and control transfers between the APU and external devices.

The Secondary Port allows the APU to directly access external devices through load and store instructions. The Secondary Port may also be used to pass information between the ATMizer Architecture and the System Controller, between two or more ATMizer Architectures, or as part of the ATMizer Architecture-to-Host Messaging System. The Secondary Port can be used to access external devices while the DMA Controller is busy, and to pass information to an external device about an active DMA operation.

In applications that require large number of Virtual Channels, the Channel Parameters can be stored in an external high-speed SRAM connected to the Secondary Port. The APU can efficiently transfer Channel Parameter Entries into the Prefetch Buffer concurrent with a DMA operation.

10.2 Operation

The Secondary Port has a multiplexed address and data bus with 4 Mbytes of address space. During the address cycle, SP_AD[31:22] contains transaction information and SP_AD[21:0] contains the physical address for the access. The address cycle is controlled externally by the SP_ASEL signal. Asserting SP_ASEL HIGH causes the L64360 to drive an address on SP_AD[31:0]. During a read operation, deasserting SP_ASEL causes the L64360 to 3-state SP_AD[31:0] so that external logic can drive the data onto SP_AD[31:0]. During a write operation, deasserting SP_ASEL causes the L64360 to drive SP_AD[31:0] with data.

The Secondary Port supports byte operations as well as dynamic bus sizing. When the Secondary Port requests a word read transaction, and the external device is byte-wide, the device can assert the $\overline{\text{SP\_BWIDE}}$ and $\overline{\text{SP\_ACK}}$ signals. When $\overline{\text{SP\_BWIDE}}$ is asserted during a word transaction, the Secondary Port generates three more transactions for the remaining three bytes (the address incremented by one each time). Dynamic bus sizing appears to the external device as a series of four single-byte read transactions. Please refer to Section 13.1, "Secondary Port," for more details.

Instruction Fetch

When the user's firmware exceeds 4 Kbytes, the L64360 cannot execute all the code from the IRAM. In this case, the user can store the extra code into the Secondary Port memory. Firmware can use either jump or branch instructions to execute code back and forth between the IRAM and the SP memory. Firmware should only use non-cacheable address space (starting at 0xA0C00000) to jump or branch to the SP memory. The latency penalty for executing an instruction off-chip is design dependent.

Single Load/Store

To acquire and update data from or to the SP memory, firmware can initiate load and store operations with an Effective Address of 0xA0CXXXXX. For Store Word, Store Halfword, or Store Byte instructions, SP_AD[31:28] reflect Write Byte Enables (active LOW) for the corresponding bytes. For load operations, the ATMizer Architecture drives SP_AD[31:28] HIGH during the address phase. Even though firmware can issue a Load Word, Load Halfword, or Load Byte instruction, the SP hardware always performs a Load Word instruction and then returns the proper bytes to the APU.

Block Fetch

When user firmware needs to acquire a block of data from the Secondary Port memory (a CPE for example), it may issue a load operation with a cacheable Effective Address of 0x00CXXXXX. Depending on how the System Control Register and the APU BCC Register are set, SP hardware will perform two-word or four-word block fetches. The fetched block must be within a two-word or four-word boundary. If the external address latch cannot support address wraparound for block fetches, it may assert SP_ASEL, which makes the SP hardware drive a new address onto SP_AD[31:0] for external logic to latch.

Byte Device Access (Boot PROM)

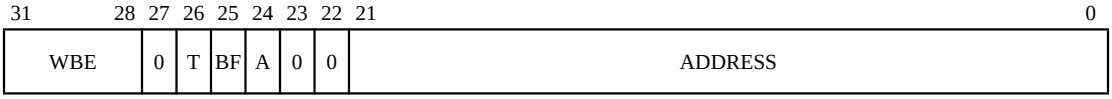
Every time the ATMizer Architecture asserts SP_RQ to initiate a word operation, SP_AD[31:0] show address and control information. The ATMizer Architecture waits until the external device asserts $\overline{\text{SP\_ACK}}$.

If the ATMizer Architecture detects the assertion of $\overline{\text{SP\_BWIDE}}$ and $\overline{\text{SP\_ACK}}$, it always expects data on SP_AD[7:0] during the data phase. After the ATMizer Architecture finishes the first byte access, it initiates three more transactions with addresses incremented by one each time. External logic has to assert $\overline{\text{SP\_BWIDE}}$ during all four accesses to ensure proper operation.

10.3 SP Address and Data Bus (SP_AD[31:0])

Figure 10.1 shows the Secondary Port Address and Data Bus, SP_AD[31:0] during the address phase (SP_ASEL asserted).

Figure 10.1
SP_AD[31:0]



WBE **Write Byte Enables** **[31:28]**
For a write operation, the ATMizer Architecture clears a WBE Bit to zero to indicate that it intends to store the corresponding byte (shown in the table below) of SP_AD[31:0] to the slave device during the data phase.

WBE	*Byte Enabled*[1]
Bit 31	SP_AD[31:24]
Bit 30	SP_AD[23:16]
Bit 29	SP_AD[15:8]
Bit 28	SP_AD[7:0]

1. If bit set to one.

For a read operation, the ATMizer Architecture sets all four bits to one since it always loads a full word.

T **Type** **26**

This bit determines the access type. The ATMizer Architecture clears this bit to zero to indicate that the Secondary Port is performing data transactions. The ATMizer Architecture sets this bit to one to indicate that the Secondary Port is fetching instructions.

BF **Block Fetch** **25**

The Secondary Port sets this bit when it requests a burst transfer. The size of the burst depends on the value programmed into the Cache Block Size in the APU Configuration Register and the System Control Register. If the beginning address is not aligned to the size of the block transfer, the Secondary Port wraps the address around to get all the words that fall within the Effective Address. If the external address latch cannot generate subsequent addresses, it may relatch the addresses by asserting SP_ASEL after each word operation is finished. For more details please refer to the functional waveforms in Section 13.1, “Secondary Port.”

A **Atomic** **24**

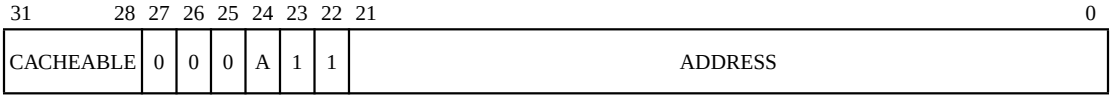
Setting this bit indicates to the external arbiter that the following atomic transactions (read-modify-writes) should be locked. It is the responsibility of the external arbiter to guarantee that the bus is not given to another master after the first read. This bit is reflected from Bit 24 of the Effective Address.

ADDRESS **Memory Address** **[21:0]**

These bits address one of the 4 Mbyte pieces of Secondary Port memory space. These bits are reflected from Bits [21:0] of the Effective Address.

Figure 10.2 shows the format of the Effective Address that the firmware uses to access memory through the Secondary Port.

Figure 10.2
SP Effective Address



CACHEABLE **Cacheable Data Access** **[31:28]**
This field distinguishes between cacheable and non-cacheable data access space.

The APU firmware clears these bits to 0x0 when the access is to a cacheable space. The SP performs a block-fetch operation and feeds the data into the four-word Prefetch Buffer. The APU firmware sets these bits to 0xA when the access is to a non-cacheable space. The SP performs a single load/store operation.

In applications that require more than 1 Kword of instructions with the extra code residing in the Secondary Port memory, the code has to be executed through non-cacheable accesses because the IRAM internal to the ATMizer Architecture is not a true cache and the Instruction Cache Controller is not activated.

A **Atomic** **24**
Setting this bit to one informs the external arbiter that the Secondary Port must retain bus ownership. This bit reflects onto Bit 24 of the Physical Address, SP_AD24, during the address cycle.

APU firmware sets and clears this bit. If the firmware issues the first load operation with the A Bit set to one, it must issue the second store operation with the A Bit set to one.

ADDRESS **Memory Address** **[21:0]**
These bits address one of the 4 Mbyte pieces of Secondary Port memory space. These bits reflect onto SP_AD[21:0], during the address cycle.

10.4 SP Hardware Design Tip

Since the SP protocol does not conform to any standard bus specification, there are several ways for an external device to handshake with the ATMizer Architecture. The following information may help the system hardware designer shorten access latency.

If there are no other masters sharing the SP device with the ATMizer Architecture, the system hardware designer can tie SP_GNT to HIGH, which makes SP_AD[31:0] have a valid address at the same cycle when SP_RQ is asserted. External latch circuitry can keep latching the address when SP_RQ is deasserted. Freezing the latch after SP_RQ is asserted makes the address available one clock cycle earlier. This method will not support atomic operation, however, since the external latch circuitry needs SP_RQ to toggle in order to get the new address.

The Write Byte Enables are valid only after SP_RQ is asserted. For systems that always perform word operations, the above method saves one clock cycle. If the firmware cannot avoid doing byte or halfword accesses, it does not help for external logic to latch the address one clock cycle earlier, since it has to wait for the valid Write Byte Enables to generate proper chip enables anyway.

Chapter 11
System Mapping

This chapter describes the ATMizer Architecture system hardware map.

This chapter has three sections:

- Section 11.1, “Memory Maps”
- Section 11.2, “Interrupts”
- Section 11.3, “Coprocessor Condition (CpCond) Connections”

11.1 Memory Maps

This section contains the following:

- Internal Memory Map
- External Memory Map
- System Memory Map Summary

All APU accesses to the internal registers and the VCR last one CPU cycle.

Chapter 14 defines all the ATMizer Architecture registers.

Internal Memory Map

Table 11.1 shows the Internal Memory Map of the APU Address Bits (Effective Addresses). The subsections following the tables explain the variable bits.

Please refer to Chapter 14 and the *CW33300 Enhanced Self-Embedding Processor Core User’s Manual* for APU Core register definitions.

Table 11.1
Internal Memory Map

APU Address Bits (Effective Address)																																
31	***30***	***29***	***28***	***27***	***26***	***25***	***24***	***23***	***22***	***21***	***20***	***19***	***18***	***17***	***16***	***15***	***14***	***13***	***12***	***11***	***10***	***9***	***8***	***7***	***6***	***5***	***4***	***3***	***2***	***1***	***0***	***R/W***	***Description***
VCR																																
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	X	X	X	X	R/W	VCR Access
PRU																																
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	0	T	0	0	0	0	0	0	0	R	Channel Group Credit Reg
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	W	Configuration Register
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	W	Stall Register
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	0	0	R	R	R	R	R	0	R	Peak Rate Pacing Counters
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0	1	1	I	0	R	R	R	R	R	0	R/W	Count Initialization Regs
ACI																																
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	R	Current Rec Cell Addr Reg
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	0	1	F	C	0	0	0	0	0	0	W	Transmit Cell Addr FIFO
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	1	0	0	0	0	0	1	1	0	0	W	Rec Cell Indicator Reg
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	1	1	1	E	0	0	0	0	0	0	0	W	Global Pacing Rate Reg
Control																																
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	R/W	System Control Register
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	0	1	1	0	0	0	0	0	0	0	0	W	Host Interrupt Register
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	1	0	0	P	0	0	0	0	0	0	0	R/W	CRC32 Register
1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	1	0	0	1	1	0	1	0	0	0	0	0	0	0	0	W	MSB Substitution Register
DMA																																
LO	LO	BC	BC	BC	BC	BC	BC	0	1	0	0	0	0	0	0	0	RD	G	0	L	L	L	L	L	L	L	L	L	L	0	0	W	DMA Control Register

PRU

Clearing the T Bit (Bit 7 of the Credit Register's Effective Address) to zero causes the PRU to clear all PRPC bits associated with CpCond2 timeout indication methods. Setting the T Bit to one causes the PRU to clear all PRPC bits associated with the Interrupt1 timeout indication method.

Clearing the I Bit (Bit 7 of the Count Initialization Register's Effective Address) to zero causes the PRU to immediately write the initialization value into both the Count Initialization Register and the Peak Rate Pacing Counter, overwriting their values. Setting the I Bit to one causes the PRU to write the initialization value into the Count Initialization Register, but the Peak Rate Pacing Counter is allowed to continue with its count. Once the count reaches zero, the PRU writes the new initialization value into the Peak Rate Pacing Counter.

Table 11.2 shows how the R Bits (Bits [5:1] of the PRPC Initialization and PRPCs' Effective Addresses) are used to select which one of the 10 Peak Rate Pacing Counters are the target of the operation.

Table 11.2 PRPC Initialization and Content Register Selection

R Bits	***Target PRPC***	***R Bits***	***Target PRPC***	***R Bits***	***Target PRPC***
00000_2	PRPC 0	00100_2	PRPC 4	10000_2	PRPC 8
00001_2	PRPC 1	00101_2	PRPC 5	10010_2	PRPC 9
00010_2	PRPC 2	00110_2	PRPC 6		
00011_2	PRPC 3	00111_2	PRPC 7		

ACI

Tables 11.3 and 11.4 show the use of the C and F Bits (Bits 6 and 7 of the Transmit Address FIFO's Effective Address).

Table 11.3 CRC10 Generation Control

C Bit	***Description***
0	Do not Generate CRC10
1	Generate CRC10

Table 11.4 HEC Error Control

F Bit	***Description***
0	Normal Operation Mode
1	If HEC is enabled, force an error into the HEC Byte

The E Bit (Bit 7 in the GPRR's Effective Address) is used to enable and disable Global Pacing. Clearing the E Bit to zero disables Global Pacing, so the ACI does not transmit any Idle Cells between Assigned Cells when the GPRR Counter reaches zero. But, if there is no cell available for transmit in the Transmit Address FIFO, ACI does transmit Idle Cells. Setting the E Bit to one enables Global Pacing, so the ACI inserts Idle Cells into the cell stream when the Global Pacing counter reaches zero.

Control

The P Bit (Bit 7 in the CRC32 Register's Effective Address) is used to specify whether the value returned in the CRC32 Register is a Partial Result or Final Result. Clearing the P Bit to zero causes the CRC32 Register to return the CRC32 Partial Result to the APU. Setting the P Bit to one causes the CRC32 Register to return the CRC32 Final Result to the APU. The Final Result is returned as the complement of the Partial Result.

DMA

Table 11.5 shows the DMA Control Register's Effective Address Field definitions. Table 11.6 shows the combined use of the RD and G Bits (Bits [14:13] of the DMA Control Register's Effective Address). For more detailed information see Section 8.2, "Registers."

Table 11.5
DMA Control Register's
Effective Address Fields

APU Address Bits	***Field***	***Description***
EA[31:30]	LO	VCR Local Address Byte Offset
EA[29:24]	BC	Transfer Length Byte Count (1 - 63, 000000_2 = 64bytes)
EA[23:15]	010000000_2	Indicates that the DMA Control Register is the target of the store
EA14	RD	Read/Write (Operation Direction)
EA13	G	Ghost Bit
EA[11:2]	L	VCR Local Address Counter

Table 11.6 Operation Direction (RD) and Ghost Bit (G) Settings

RD Bit	*G Bit*	*Description*
0	0	Initiate DMA Write Operation
0	1	Initiate DMA Ghost Write Operation
1	0	Initiate DMA Read Operation
1	1	Illegal Combination

External Memory Map

Table 11.7 shows the External Memory Map (Secondary Port and Host/ DMA Port Direct Access Memory Map).

Table 11.7
External Memory Map

APU Address Bits (Effective Address)																												*Description*
31	*30*	*29*	*28*	*27*	*26*	*25*	*24*	*23*	*22*	*21*	*20*	*19*	*18*	*17*	*16*	*15*	*14*	*13*	*12*	*11*	*10*	*9*	*8*	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*	
Cacheable - User/Supervisor																																
0	0	0	0	0	0	0	A[1]	1	0	X[2]	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	Host/DMA Port Direct Access
0	0	0	0	0	0	0	A	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	Secondary Port Direct Access
Non-Cacheable - Supervisor																																
1	0	1	0	0	0	0	A	1	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	Host/DMA Port Direct Access
1	0	1	0	0	0	0	A	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	Secondary Port Direct Access

1. If A = 1, the operation is atomic. If A = 0, it is not atomic.
2. X can be either zero or one.

System Memory Map Summary

Table 11.8 shows the system memory map summary.

Table 11.8 System Memory Map Summary

Destination	***APU Effective Address***
Host/DMA Port Cacheable Direct Access	0x00800000 to 0x00BFFFFC
Secondary Port Cacheable Direct Access	0x00C00000 to 0x00FFFFFC
Host/DMA Port Non-cacheable Direct Access	0xA0800000 to 0xA0BFFFFC
Secondary Port Non-cacheable Direct Access	0xA0C00000 to 0xA0FFFFFC
Host/DMA Port Atomic Access	0xA1800000 to 0xA1BFFFFC
Secondary Port Atomic Access	0xA1C00000 to 0xA1FFFFFC
VCR Access	0xFFF00000 to 0xFFF00FFC
PRU Channel Group Credit Register	0xFFF04000 or 0xFFF04080
PRU Configuration Register	0xFFF04100
PRU Stall Register	0xFFF04200
PRU Count Initialization Register 0	0xFFF04300 or 0xFFF04380
PRU Count Initialization Register 1	0xFFF04302 or 0xFFF04382
PRU Count Initialization Register 2	0xFFF04304 or 0xFFF04384
PRU Count Initialization Register 3	0xFFF04306 or 0xFFF04386
PRU Count Initialization Register 4	0xFFF04308 or 0xFFF04386
PRU Count Initialization Register 5	0xFFF0430A or 0xFFF0438A
PRU Count Initialization Register 6	0xFFF0430C or 0xFFF0438C
PRU Count Initialization Register 7	0xFFF0430E or 0xFFF0438E
PRU Count Initialization Register 8	0xFFF04320 or 0xFFF043A0
PRU Count Initialization Register 9	0xFFF04324 or 0xFFF043A4
PRU Peak Rate Pacing Counter	0xFFF04300 to 0xFFF04324
ACI Current Received Cell Address Register	0xFFF04400
ACI Transmit Cell Address FIFO	0xFFF04500, 0xFFF04540, 0xFFF04580, or 0xFFF045C0
ACI Received Cell Indicator Register (Decode Circuit)	0xFFF0460C
ACI Global Pacing Rate Register	0xFFF04700 or 0xFFF04780
System Control Register	0xFFF04A00

(Sheet 1 of 2)

Table 11.8 (Cont.) System Memory Map Summary

Destination	*APU Effective Address*
Host Interrupt Register (Decode Circuit)	0xFFF04B00
CRC32 Register	0xFFF04C00 or 0xFFF04C80
MSB Substitution Register	0xFFF04D00
DMA Control Register	0xXX40XXXX[1]
APU BIU/Cache Configuration Register	0xFFFE0130
APU Coprocessor Registers	$3, $5, $6, $7, $8, $9, $11, $12, $13, $14, $15

(Sheet 2 of 2)

1. X can be zero or one.

11.2 Interrupts

The APU has six interrupt inputs. Each of these interrupts can be enabled or disabled by software running on the APU (see the *CW33300 Enhanced Self-Embedding Processor Core User's Manual*). The ATMizer Architecture uses all six of the APU interrupt pins. User firmware may choose to enable and use any of these interrupts.

Table 11.9 defines the six APU interrupts.

Table 11.9 APU Interrupts

Interrupt	*Definition*
0	Watchdog Timeout
1	PRU Counter Timeout
2	GPINT_AUTO (General Purpose Interrupt Available on the Pin)
3	DMA Complete
4	One Address Left
5	Received Cell Buffer Half Full

11.3 Coprocessor Condition (CpCond) Connections

The APU can check the state of any one of the CpCond inputs by executing a Branch on Coprocessor X Condition True/False instruction. GPINT_TST differs from GPINT_AUTO in that it is not an interrupt in the classic sense but simply a signal whose state can be tested by the APU by issuing the Branch on CpCond0 True/False instruction.

Table 11.10 defines the four coprocessor conditions.

Table 11.10
CpCond Definitions

CpCond	***Definition***
0	GPINT_TEST (General Purpose Test Pin - Pin 170) (see Section 4.7, “ATMizer Architecture-to-Host Messaging”)
1	Received Cell Indication (see Section 9.5, “ACI Receiver”)
2	PRU Transmit Request (CGCR Bit Set) (see Chapter 7)
3	DMA Busy (see Section 8.8, “DMA Operation Completion”)

Chapter 12
Operation

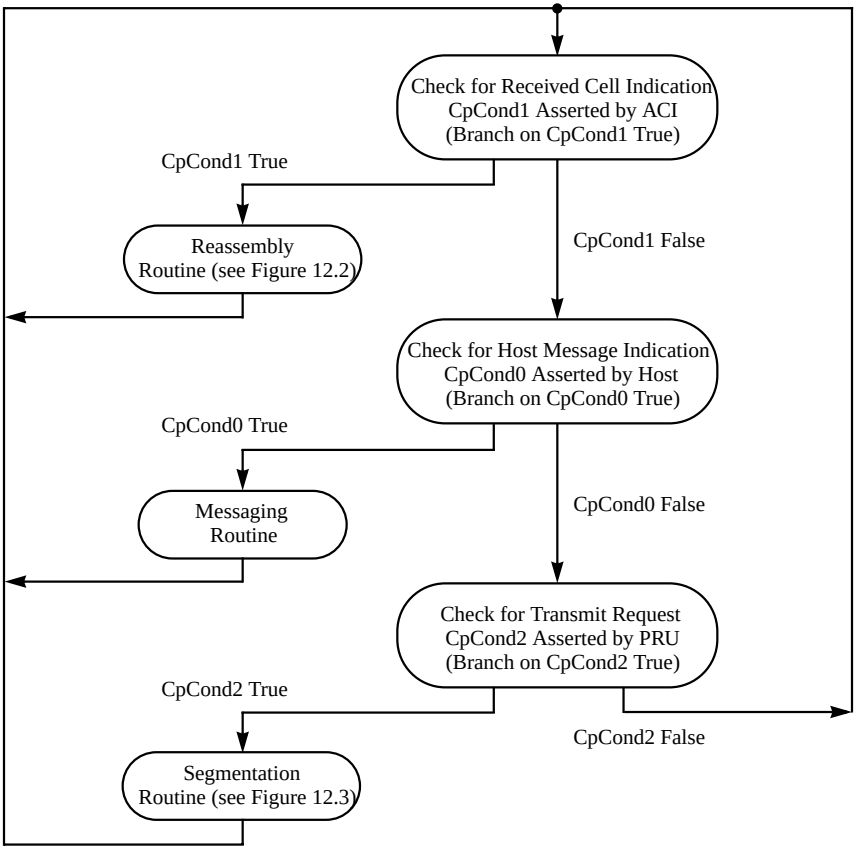
This chapter describes the ATMizer Architecture operation.

This chapter has six sections:

12.1 Programming the ATMizer Architecture

No two ATMizer Architecture applications are likely to be identical. Each system designer usually creates APU firmware specifically tailored to his/her implementation. The basic firmware routine usually centers around the Idle Loop shown in Figure 12.1, which checks for the existence of one of three conditions and takes the appropriate actions if one of the states exists. The order in which the CpCond pins are tested in our example Idle Routine is significant. In this example, the firmware checks for Received Cell Indication first since it is more important not to drop a received cell (if the received cell buffer overflows) than it is to prevent an Idle Cell from being transmitted.

Figure 12.1
APU Idle Loop



The user firmware should always check for Received Cell Indication before checking for either a Host Messaging Request or a Transmit Cell Request. Because the ATMizer Architecture can be asked to accomplish certain complex functions, the servicing of either a Received Cell Indication, transmit cell request, or message request could take longer than the time normally allotted in steady state operation. As a result, cells may accumulate in the VCR and firmware may wish to always drain this buffer before proceeding with the segmentation routine.

12.2 Theory of Operation

This section contains two subsections that explain how the ATMizer Architecture operates during cell reassembly and segmentation.

Reassembly

When a cell arrives, the ACI extracts and checks the HEC Field and puts the cell into the VCR. Since there is no way of detecting the AAL type yet, the CRC10 circuitry must check the CRC10 Field of every incoming cell regardless of the AAL type. Once the cell is buffered, the ACI generates an interrupt to notify the APU that there is at least one new cell in the buffer. Alternately, the APU can poll a condition signal to find out if there is at least one new cell in the buffer.

Firmware can implement an interrupt mechanism using the Received Cell Indicator Register (RCIR) as an up/down counter. Each time a new cell arrives, the counter is incremented by one. If the counter is greater than zero, the ACI asserts interrupts and coprocessor condition signals. Each time the APU processes a cell, at the end of the reassembly routine the APU must write to the RCIR to decrement the counter by one. The first step in processing a cell is to extract the cell header and find the VCI and VPI fields. The VCI/VPI are used to index a data structure that describes the cell.

Once the data structure is retrieved, the APU can find out more information about how to proceed with processing the cell. The cell's data structure (called a Channel Parameter Entry or CPE) may contain information such as the AAL type, the Host memory address (where the CS-PDU resides), the CRC32 Partial Value for AAL 5, and the previous SAR Header for AAL 3/4. Next, the CPU may decide whether to translate the VCI/VPI and switch the cell back for transmission or to terminate the cell. In the case of termination, the APU programs the DMA controller with the Host memory address found in the data structure. In the case of AAL 5, the APU initializes the CRC32 generation circuitry with the CRC32 Partial Value found in the data structure.

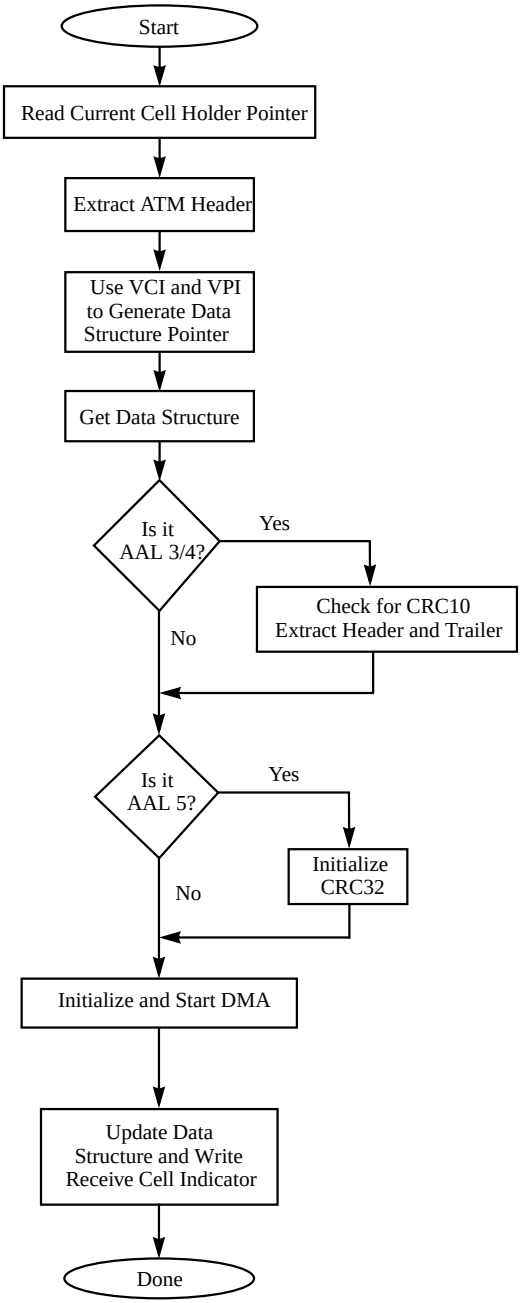
For AAL 5, it is necessary to have partial CRC32 value because the CRC32 mechanism is calculated over the whole CS-PDU. Once the SAR-SDU is transferred into the host memory, the CPU retrieves the new CRC32 Partial Value and updates the data structure. If this is the last SAR-SDU for AAL 5, the CRC32 value can be checked against the last four bytes of the payload. In the case of an error, the APU informs the Host by some predefined messaging protocol. If it is not the last SAR-SDU for the particular CS-PDU, the APU also updates the Host memory address for the next SAR-SDU transfer.

Another significant advantage in having an embedded APU is the ability to allocate the Host memory dynamically through a predefined messaging protocol. Dynamic memory allocation is very attractive for the AAL 5 reassembly process because there is no way to know the CS-PDU size beforehand. For AAL 3/4, there is a Buffer Size Field in the CS-PDU Header. This field does not exist for AAL 5 so the user information can be up to 65536 bytes. If 1 Kbyte of CS-PDU is active and 64 Kbytes of memory space is allocated for each active CS-PDU, then 64 Kbytes of memory needs to be allocated beforehand. If the average packet size is 4 Kbytes, then almost 94% of memory space is wasted (60 Mbytes).

Another attractive feature of having an embedded APU and DMA controller is that the CS-PDU needs not be contiguous in the Host memory. The CS-PDU can be scattered throughout the Host memory. This feature is extremely attractive in desktop applications such as ATM adapter cards. Most operating systems configure and allocate memory in pages and use virtual addressing. The page size is usually small. For example, in the SPARC Reference Memory Management Unit (MMU), one page is 4 Kbytes. A 64-Kbyte AAL 5 CS-PDU needs 16 pages. When the MMU allocates these pages, most likely they will not be contiguous. After a CS-PDU is assembled, the embedded CPU can interrupt and notify the Host processor through a predefined messaging system.

Figure 12.2 illustrates a typical reassembly routine triggered by the interrupt or condition signals.

Figure 12.2
Reassembly Routine



Segmentation

The event that triggers a segmentation process can be generated when one or more of the Peak Rate Pacing Counters elapse or by a predefined messaging system with the Host processor. For AAL 1 datastreams, the APU can poll a memory map register for a Ready to Segment Flag to be set by an external device such as a DS1 termination device. In this discussion, the Peak Rate Pacing Counters are used as the segmentation triggering mechanism.

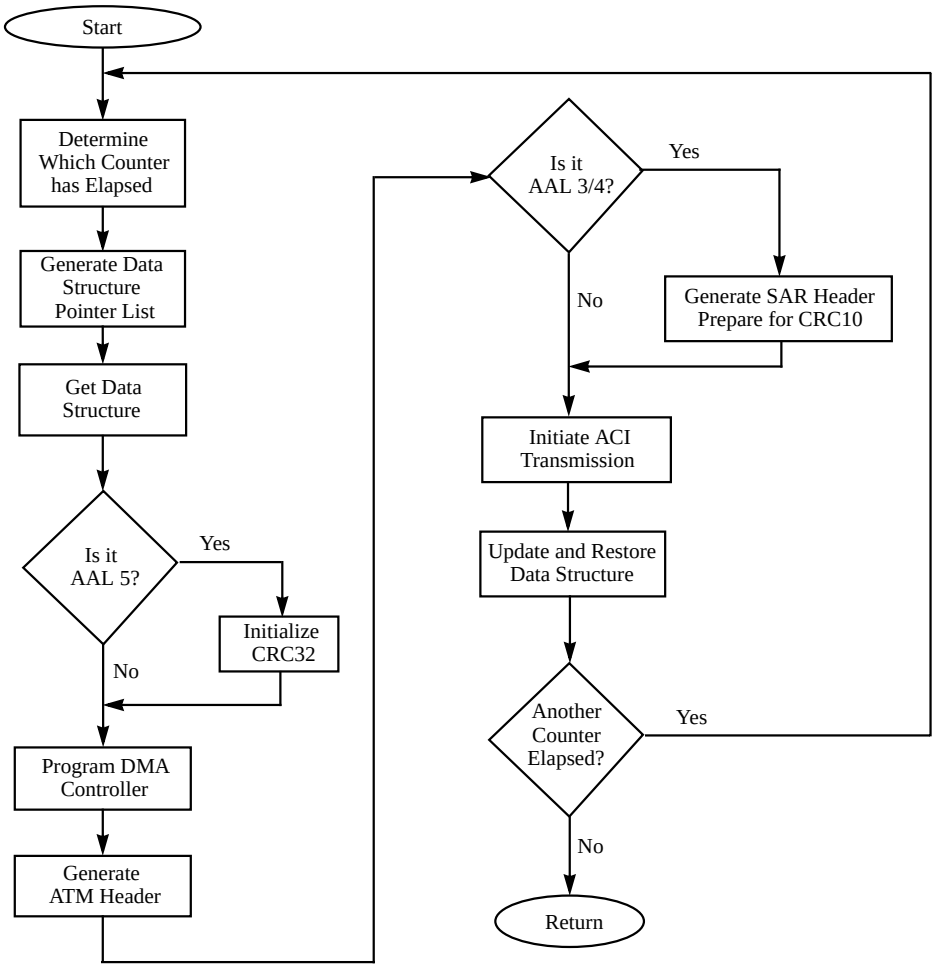
The Peak Rate Pacing Counters can be used to index a link list of pointers. These pointers point to a data structure describing the particular CS-PDU to be segmented. When one or more counters elapse, the APU can be interrupted and the firmware should vector into the segmentation routine. The first task for the segmentation routine is to use the counter and index a particular link list. The APU can implement a priority mechanism to select which CS-PDU will be segmented first.

The data structure to describe a CS-PDU for segmentation process is slightly different compared to the data structure for the reassembly process described in the previous subsection. The data structure for segmentation contains information such as the ATM Header, the CRC32 Partial Value for AAL 5, the current Host memory address pointing to the SAR-SDU to be transferred, and the transfer count. Other useful control information such as the maximum burst size and priority can be included in the data structure. The maximum burst size can be used to inform the APU to transfer multiple SAR-SDUs from the Host memory into the cell buffer memory for segmentation.

For AAL 5 CS-PDU, the CPU needs to initialize the CRC32 circuitry with the CRC32 Partial Value before the transfer begins. Once the SAR-SDU is in the cell buffer memory, the APU needs to generate the cell header. The cell header is retrieved from the data structure and the APU may process some of the fields, such as the VCI, the VPI, the GFC, the CLP, and the PT. For AAL 3/4, the APU must also generate the SAR Header and Trailer. The SAR Header and Trailer from the previous SAR-SDU can also be stored in the data structure. The APU may need to manipulate some of the field such as incrementing the SN Field. When the cell is ready for transmission, the APU informs the ACI to fetch the cell from the cell buffer memory and passes it to the TCS. The HEC can be inserted dynamically during the transmission. Depending on the physical layer used, the HEC generation circuitry is programmable and is capable to insert or skip the HEC Field.

If the cell originated from AAL 3/4 CS-PDU, the APU activates the CRC10 circuitry and the CRC10 Field is inserted at the end of the cell during transmission. Before returning from the segmentation routine, the APU has to update the data structure by restoring the new CRC32 Partial Value for AAL 5, updating the SAR header for AAL 3/4, incrementing the transfer count, and so on. Figure 12.3 illustrates a typical segmentation routine triggered by the peak rate counters. For more information on CRC10 generation for AAL 3/4, refer to Section 9.8, "CRC10 Generation and Error Checking."

Figure 12.3
Segmentation Routine



12.3 Initializing the PRU

At the beginning of operation, software should initialize the PRU. The recommended method for initializing the PRU is as follows:

1. Software stalls all timers by writing 0x3FF into the Stall Register.
2. Software programs the configuration register to indicate to the PRU which clock each of the PRPCs is to be driven from (Configuration Register Bits [9:0]) and to indicate to the PRU which time out indication method is to be used for each of the three timer groups (Bits [12:10]).
3. Software clears the CGCR by reading the CGCR twice. The first time the CGCR should be read with Address Bit 7 cleared to zero to clear those bits associated with the CpCond2 timeout method. The second time the CGCR should be read with Address Bit 7 set to one to clear those bits associated with the Interrupt1 reporting mechanism.
4. Software programs the initialization value for each of the PRPCs and enables the counter by writing to the Stall Register, which unstall the PRPCs.

The following code demonstrates how to initialize and use the PRU:

```
#include "regdef.h"
/*********************************************************
This program is written to test the ATMizer Architecture PRU.

The following software checks that:
1) The Credit Register works properly with Interrupt1
2) The clearing mechanism for timers associated with
Interrupt1 works properly
*********************************************************/

/* Define Register Addresses */

#define Magic       0xa0ffff00
#define Happy       0xa0ffff04
#define Sad         0xa0ffff44

#define cir_prpc    r1
#define conf_reg    r2
#define stall_reg   r3
#define cr_reg      r4
```

```
#define apu_bcc     r5
#define hbs_addr    r6
#define sys_ct_reg  r7
#define apu_d       r8
#define stall_val   r9
#define imm_val     r12
#define del_val     r13

.text
.set    noreorder
.set    noat

/* Initialize On-chip and Off-chip Addresses */

la      r30, Happy
la      r29, Sad

/*********************************************************
r28, r27, r26 are reserved for address of exception subroutine
*********************************************************/
la      r25, excp1
la      r26, excp2
la      r27, excp3
la      r28, excp4

la cir_prpc, 0xfff04300    # count initialization reg_prpc

la conf_reg, 0xfff04100
la stall_reg, 0xfff04200
la cr_reg, 0xfff04000

la apu_bcc, 0xfffe0130
la hbs_addr, 0xa080ae00
la sys_ct_reg, 0xfff04a00

li apu_d, 0x00004890       # apu_reg <- nopad, enable
                           # i$, d$, 4-word block
li stall_val, 0xffffffff
li imm_val, 0x004          # Timer starting count value
li del_val, 0xffe          # The value loaded to CIR only
                           # (delays load)
```

```
/*********************************************************
Clear the error condition bits in the sys_ct_reg
set block size = 4 (for hp block fetches by CPU)
*********************************************************/
li r10, 0x00080070
sw r10, (sys_ct_reg)

/*********************************************************
Initialize APU BCC Reg: no wait state between bus
transactions, enable i$, d$, block size = 4
*********************************************************/
sw apu_d, (apu_bcc)

/* Clear PRU CpCond and Interrupt */

sw stall_val, (stall_reg)  # Write 0xfffffff to stall reg
sw r0,  (conf_reg)         # Initialize PRU Config Reg with r0
lw r10, (cr_reg)           # Read from Credit Reg
lw r10, 0x80(cr_reg)       # Read from Credit Reg

/* Enable interrupt.1 */

lw r11, (apu_bcc)          # Read APU BCC Register
nop
ori r11, r11,0x00001000
sw r11, (apu_bcc)          # Write to BCC Reg to change
                           # interrupt polarity
nop
li r11, 0xf0400801         # Write to cp0 Status Reg, bev = 1,
                           # int.1 = 1, en-intr
mtc0 r11, $12
nop

/*********************************************************
Verify that the credit register is functioning properly
This is done for each CIR/PRPC timer using Interrupt.1
reporting Mechanism.
*********************************************************/

/* Disable interrupts */
```

```
li r11, 0xf0400800         # Write to cp0 Status Reg, bev = 1,
                           # int.1 = 1, dis-intr
mtc0 r11, $12
nop

/* Stall the timers */

sw stall_val, (stall_reg)  # Write 0xfffffff to Stall Reg to
                           # stall all the timers

/* Configure Timers 0,1,2,3 as interrupt1 Timers */

li r13, 0x0400             # Configure Timers 0,1,2,3 as
                           # intr.1 (Bit 10 = 1)
sw r13, (conf_reg)

/*******************************************************
Program Cpcond2 timers with immediate and delayed values
*******************************************************/

li imm_val, 0x04           # Initial value loaded to Cpcond2
                           # timers
li del_val, 0x20           # Delayed timer value to Cpcond2
                           # timers

sh imm_val, 0x8(cir_prpc)  # Load cir/timer4 immed. A7 = 0
sh imm_val, 0xa(cir_prpc)  # Load cir/timer5 immed. A7 = 0
sh imm_val, 0xc(cir_prpc)  # Load cir/timer6 immed. A7 = 0
sh imm_val, 0xe(cir_prpc)  # Load cir/timer7 immed. A7 = 0

sw imm_val, 0x20(cir_prpc) # Load cir/timer8 immed. A7 = 0
sw imm_val, 0x24(cir_prpc) # Load cir/timer9 immed. A7 = 0

/*******************************************************
Load delayed timer values into Cpcond2 timers
*******************************************************/
sh del_val, 0x88(cir_prpc) # Load cir/timer4 with delayed
                           # value A7 = 1
sh del_val, 0x8a(cir_prpc) # Load cir/timer5 with delayed
                           # value A7 = 1
```

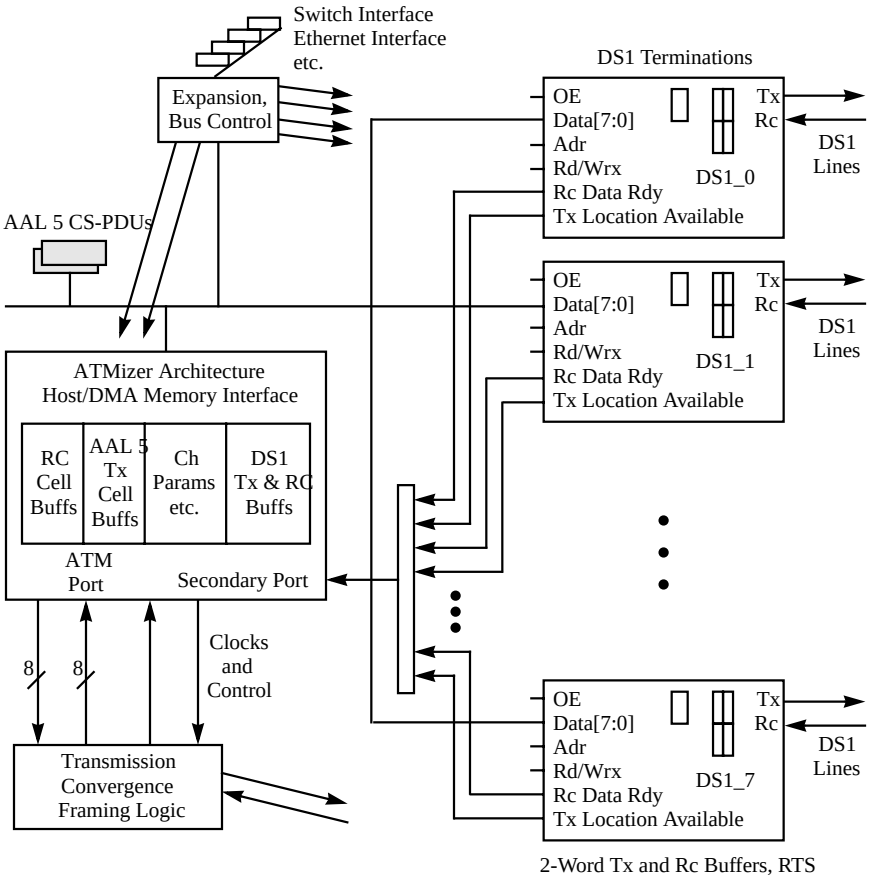
```
sh del_val, 0x8c(cir_prpc) # Load cir/timer6 with delayed
                           # value A7 = 1
sh del_val, 0x8e(cir_prpc) # Load cir/timer7 with delayed
value A7 =1

sw del_val, 0xa0(cir_prpc) # Load cir/timer8 with delayed
                           # value A7 = 1
sw del_val, 0xa4(cir_prpc) # Load cir/timer9 with delayed
                           # value A7 = 1

/*********************************************************
Program interrupt.1 timers with immediate and delayed values
*********************************************************/

li imm_val, 0x2f           # Initial value loaded to
                           # interrupt.1 timer0
sh imm_val, 0x0(cir_prpc)  # Load cir/timer0 immed. A7 = 0
li imm_val, 0x8f           # Initial value loaded to
                           # interrupt.1 timer1
sh imm_val, 0x2(cir_prpc)  # Load cir/timer1 immed. A7 = 0
li imm_val, 0x10f          # Initial value loaded to
                           # interrupt.1 timer2
sh imm_val, 0x4(cir_prpc)  # Load cir/timer2 immed. A7 = 0
li imm_val, 0x1af          # Initial value loaded to
                           # interrupt.1 timer3
sh imm_val, 0x6(cir_prpc)  # Load cir/timer3 immed. A7 = 0

/*********************************************************
Load delayed timer values into interrupt.1 timers
*********************************************************/
li del_val, 0x3ff          # Delayed timer value to
                           # interrupt.1 timers
sh del_val, 0x80(cir_prpc) # Load cir/timer0 with delayed
                           # value A7 = 1
sh del_val, 0x82(cir_prpc) # Load cir/timer1 with delayed
                           # value A7 = 1
sh del_val, 0x84(cir_prpc) # Load cir/timer2 with delayed
                           # value A7 = 1
sh del_val, 0x86(cir_prpc) # Load cir/timer3 with delayed
                           # value A7 = 1
```

```
/*********************************************************
Clear the Credit Register
*********************************************************/

lw r10, (cr_reg)           # Read from Cpcond2 address. A7 = 0
lw r10, 0x80(cr_reg)       # Read from interrupt1 address.
                           # A7 = 1

/*********************************************************
Enable interrupt.1
*********************************************************/
li      r11, 0xf0400801    # Write to cp0 status reg, bev = 1,
                           # int.1 = 1, en-intr
mtc0 r11, $12
nop

/*******************************************************
Release all the timers
*******************************************************/
sw r0, (stall_reg)         # Release all timers
```

12.4 The ATMizer Architecture in Operation

Figure 12.4 shows the ATMizer Architecture in a system supporting AAL 1 and AAL 5 circuit termination and cell switching.

Figure 12.4 ATMizer Architecture Example



Data Types Supported

The ATMizer Architecture is capable of handling a combination of data types from a variety of data sources. If the necessary data and control information, such as the Residual Time Stamp (RTS) values for AAL 1 connections, can be accessed by the ATMizer Architecture from a memory mapped entity (either a RAM or a peripheral interface controller), the ATMizer Architecture can create an ATM cell or cell stream from the data source. Realtime datastream sources can be Digital Signal Level 1 (DS1), line termination, or packet generating sources such as workstations, packet-based LANs, and WAN interfaces. The data source can also be ATM cells switched from other ATM ports which is mapped into the ATMizer Architecture DMA memory space or is accessible over the ATMizer Architecture ATM Port Cell Interface.

Cell Generation

Almost all aspects of the cell generation process are controlled by the ATM Processing Unit (APU) under user firmware control. To accomplish segmentation, the APU functions as an event-driven device. A segmentation triggering event can be an external event, such as the filling of a DS1 buffer or an internal event such as the timeout of one of the ten internal Peak Rate Pacing Counters (PRPCs).

The APU detects external events by periodically polling a Host/DMA Port or Secondary Port memory-mapped register that has a bit associated with each possible external event, or by polling the GPINT_TST signal for an indication that an external event has occurred. Polling GPINT_TST is a faster mechanism because its state can be tested with a single APU instruction (Branch on Coprocessor Condition 0 True). However, since GPINT_TST may be used as part of the Host-to-ATMizer messaging system, the assertion of GPINT_TST may have to be qualified by access to a message-type field or register somewhere in the Host/DMA Port or Secondary Port memory space. For the example in Figure 12.4, GPINT_TST assertion might alert the ATMizer Architecture to a DS1 Buffer Full Condition.

In general, internal PRPC timeout events are used to pace the segmentation of CS-PDUs while external events (GPINT_TST assertion) are used to pace cell generation from realtime datastreams. When one or more counters times out the Pacing Rate Unit responds by asserting the Coprocessor Condition 2 (CpCond2) input to the APU. The APU frequently checks the state of this input by executing a single Branch on Coprocessor Condition 2 True instruction. If the APU detects CpCond2 asserted, it branches to the segmentation routine and reads in a 10-bit value from the Channel Group Credit Register (CGCR) that indicates which counters have expired. The APU can then proceed to segment the CS-PDUs associated with the Peak Rate Pacing Counters that have expired. Since the APU can read the CGCR at any time, even in the midst of servicing a Channel Group, the user is able to implement almost any channel priority scheme that fits the application.

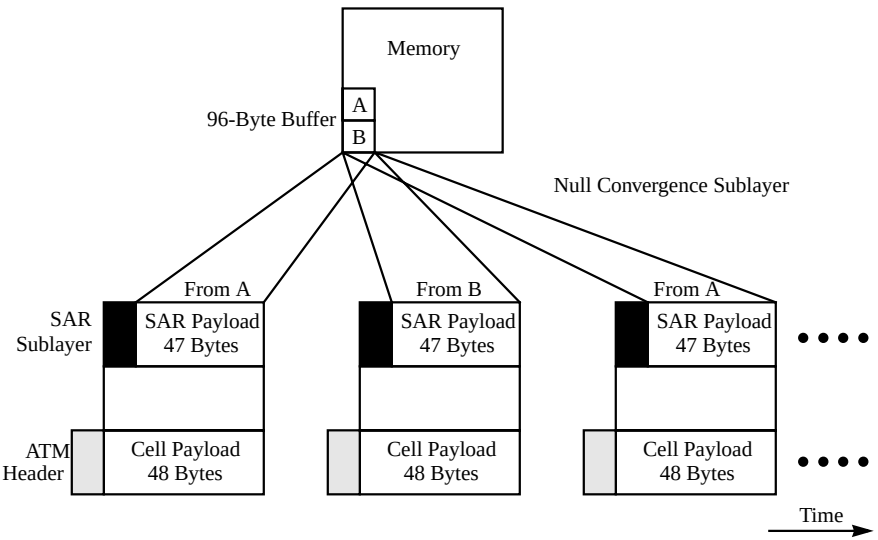
AAL 1 Realtime Datastreams

The ATMizer Architecture is capable of generating AAL 1 SAR-PDUs from memory-mapped data buffers. In most cases, datastreams such as DS1 lines will be terminated and synchronized to the ATMizer System Clock, then prebuffered in dual 48-byte buffers in main memory. Once a

buffer fills, the ATMizer Architecture can be instructed (through an external event) to retrieve the SAR User Payload, retrieve RTS, or generate Sequence Number/Sequence Number Protection (SN/SNP) values and append the SAR Header and transfer the cell to the transmission convergence framing logic using the ACI Transmitter. As the ATMizer Architecture is generating a cell from one buffer, the other buffer is being refilled by the realtime data source. Eventually the second buffer fills and the first buffer becomes the active fill buffer. AAL 1 datastreams are continuous in time. The APU under user firmware control creates the Sequence Number and Sequence Number Protection fields internally but is passed the Residual Time Stamp Field from an external device. Residual Time Stamp values can be passed to the ATMizer Architecture in Byte 0 of the SAR-SDU (the logical positioning, in this case external logic calculates the RTS and writes it into the data buffer) or the APU can proactively retrieve the RTS value, when needed, utilizing either the Secondary Port or the DMA Controller. The APU implements RTS and SN/SNP interleaving on transmission and passes the SAR SDUs and RTS values to the appropriate buffers or interfaces. The APU also performs SN/SNP checking on reassembly. The actual AAL 1 cell generation and received cell handling routines must be written by the user.

Figure 12.5 shows AAL 1 Circuit Emulation and Data Buffering.

Figure 12.5 AAL 1 Circuit Emulation and Data Buffering



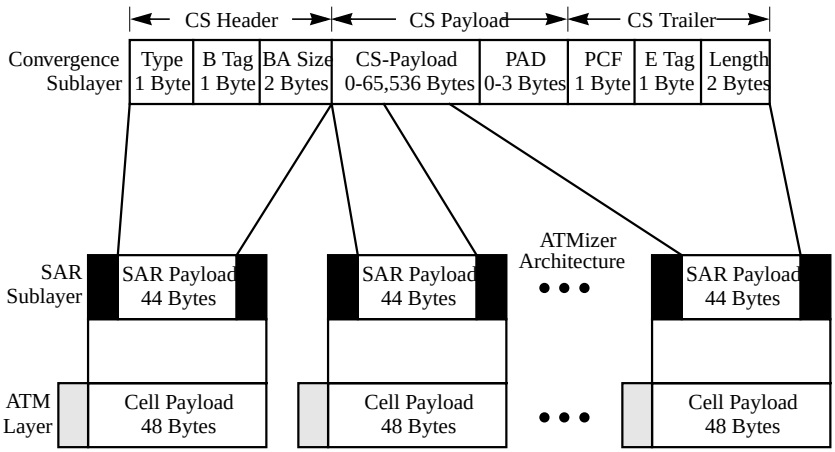
In situations where DS1s are to be sourced over an ATM port, the DS1 low data rates allow for multiple lines to be easily handled. In low-speed

applications, the ATMizer Architecture itself can be programmed to handle the transfer of data from a small-word buffer in the DS1 Physical Interface Device to the dual 48-byte buffers in main memory. In some applications the VCR itself could provide the dual data buffer functionality. Using the ATMizer Architecture in this way alleviates the need for intelligent DMA operations at the DS1-Main Memory Interface and simplifies memory controller design. Since the overhead on the ATMizer Architecture to facilitate these transfers is quite high, dumb DS1 ports may only be usable at ATM port speeds at or below DS3 rates. It is up to the user to make a final determination if a chosen implementation can sustain the desired throughput rates.

AAL 3/4 and 5 CS-PDUs Segmentation

Figure 12.6 shows AAL 3/4 CS-PDU Segmentation.

Figure 12.6
AAL 3/4 CS-PDU Segmentation

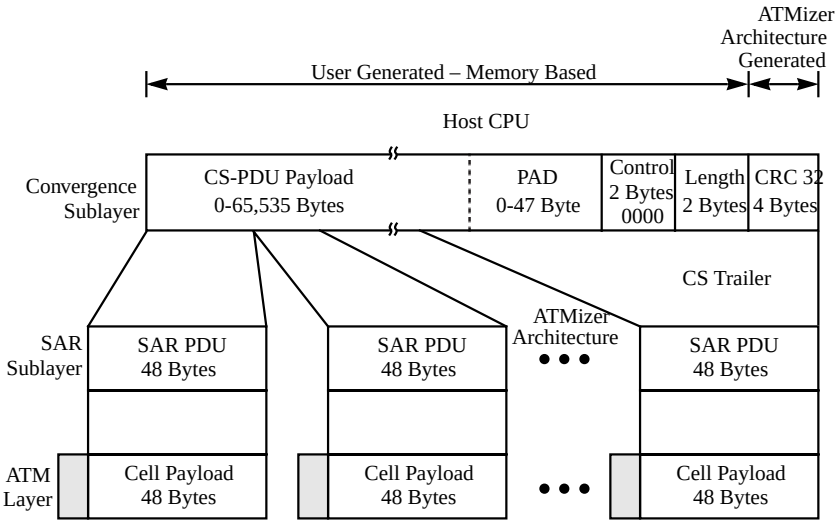


If an internal event occurs (a PRPC has expired forcing the assertion of CpCond2), the APU firmware can determine which PRPC expired by reading the Channel Group Credit Register. The APU firmware can then parse through the list of Channel Parameter Entries that are attached to the expired PRPC, segmenting a number of cells from each CS-PDU before proceeding on to the next entry in the Channel Group. As the APU parses through the Channel Parameter Entries in the Channel Group, it can generate one or more cells from a given CS-PDU before proceeding on to the next Channel Parameter Entry in the list. This list will either be in the VCR or main memory, depending on the application. VCR-resident lists have limits on their sizes (a limit on the number of channels that can be active

simultaneously) but allow for less costly memory system implementations, while memory-based lists have few restrictions on their size but may require a fast SRAM to support the processors need for fast access to the entry (as well as fast access to restore the updated entry to memory at the end of the segmentation/cell generation burst for each Channel Parameter Entry/CS-PDU).

Figure 12.7 shows AAL 5 CS-PDU Segmentation.

Figure 12.7
AAL 5 CS-PDU Segmentation



Contiguous CS-PDUs

In the most straightforward of system implementations, AAL 3/4 and 5 CS-PDUs are created in system memory by a Host Processor. The ATMizer Architecture's job is to segment these CS-PDUs into a series of SAR-SDUs, generate and append ATM Adaptation Layer headers and trailers and ATM headers to the SAR-SDUs, and then transfer the newly built cells to the external Transmission Convergence Sublayer framing logic one byte at a time using the ACI Transmitter. CS-PDUs undergoing segmentation are resident and contiguous in system memory prior to the ATMizer Architecture beginning the segmentation process. In addition to performing segmentation and ATM cell generation, the ATMizer Architecture will also calculate the CRC32 for AAL 5 CS-PDUs and append the resulting four bytes of CRC32 code to the end (Bytes [53:50]) of the last cell generated from the given AAL 5 CS-PDU. The Host Processor constructs the entire AAL 3/4 or 5 CS-PDU in system memory but, in the

case of AAL 5, should stuff all zeros into the last four bytes (the CRC32 Field).

Non-Contiguous CS-PDUs

This subsection describes the Gather function of Scatter-Gather DMA.

In more complicated system environments, CS-PDUs may be resident in noncontiguous memory. Memory fragmentation may occur in ATM network interface card applications, for instance, if the operating system builds higher-layer header fields apart from the actual user payload portion of the packet, or if the operating system creates headers from different layers physically separate though they logically belonging to the same CS-PDU. It may also occur if the User Payload Field consumes more than one page in a virtual memory system and memory management software allocates noncontiguous pages to the application. Forced moves to create a contiguous CS-PDU are wasteful of system resources and time. Fortunately, such moves are unnecessary in systems employing the ATMizer Architecture.

In routing applications (or CSU/DSU applications), the system designer may wish to provide for the segmentation of packets (CS-SDUs) prior to their complete arrival. Segmenting a CS-SDU as it arrives reduces the amount of buffer memory required in the bridging mechanism. It also reduces the latency attributable to the router. In applications employing ATM Adaptation Layer 5, the ATMizer Architecture can begin packet segmentation as soon as enough bytes arrive for the Host Processor to establish the route and before the Host Processor has built the CS-PDU trailer.

In addition, the router may allocate memory to incoming packets in blocks of size less than the maximum packet size (these blocks are referred to as *memory fragments*). This router memory allocation mechanism is useful in applications where packet sizes can vary dramatically. Small packets may take up a single memory fragment while much larger packets may require the allocations of several fragments. The ATMizer Architecture proceeds through the segmentation process one fragment at a time; communicating with the Host Processor or accessing a link list of the CS-PDU from system memory as fragment boundaries are reached.

In Gather applications, the APU periodically reaches the end of a particular CS-PDU fragment. The APU must be able to determine if it has reached the end of a fragment or if it has actually reached the end of the CS-PDU.

This information is needed to ensure that the APU does not prematurely insert an EOM identifier into the SAR Headers of AAL 2 and 3/4 cells or encode an EOM identifier into the PTI fields of the ATM Headers of AAL 5 cells. Therefore, it is important that a flag field be included in the Channel Parameter Entry that indicates whether the fragment represents end of CS-PDU or if more fragments exist for the CS-PDU. Firmware running on the APU must check this condition during the segmentation process. Since the APU must check the resulting byte count each time it decrements it, it is possible to signal end-of-CS-PDU by providing a byte count in the Channel Parameter Entry that will reach exactly zero at the end of a fragment that represents the end of the CS-PDU and one that will produce a negative result for fragments that are not the last fragment (the byte count would be at least one byte less than the actual count). These and other techniques can be employed to dramatically reduce the number of APU instructions required to generate (or process) a cell and shall be expanded upon later in the section on programming the APU.

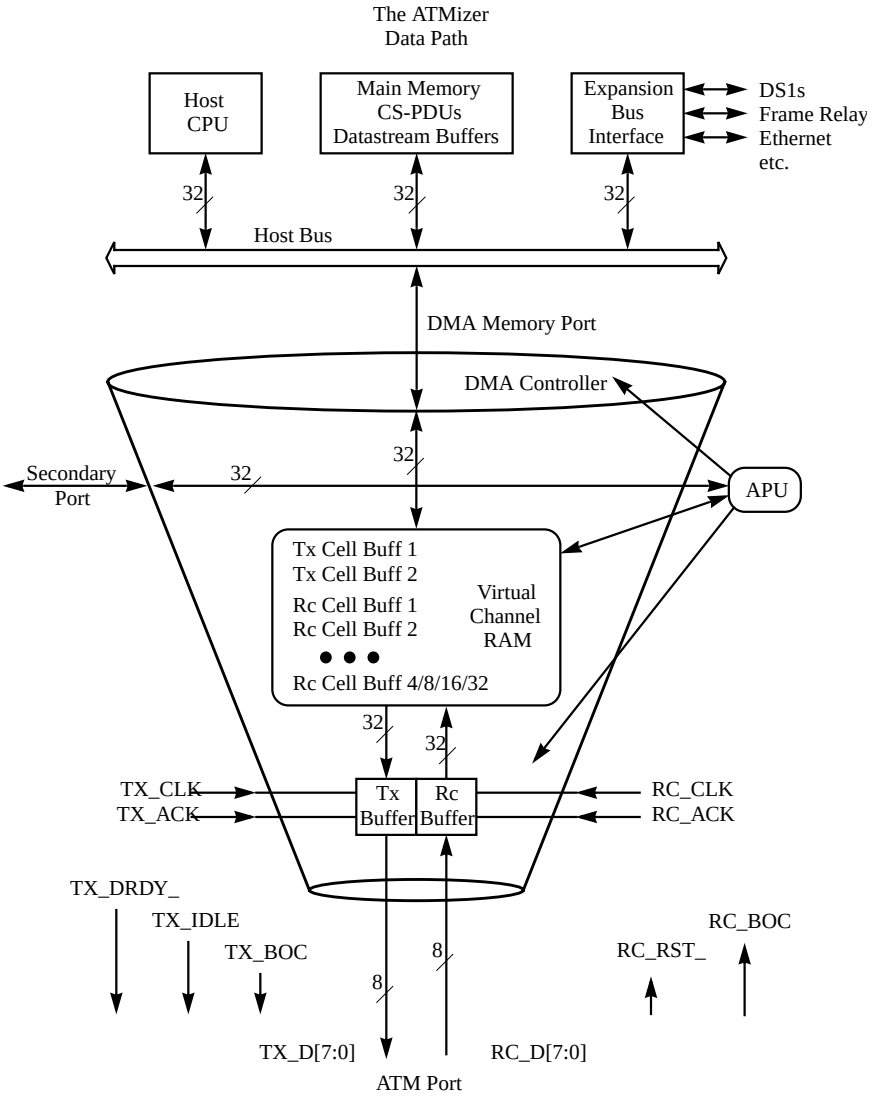
As mentioned previously, the APU may choose to generate more than one cell from a given CS-PDU before proceeding on to the next CS-PDU. This choice is up to the user but it is important to understand that generating multiple cells per CS-PDU reduces the number of APU cycles required to build a cell. The APU cycles required to retrieve and restore the Channel Parameter Entry for the CS-PDU can be amortized over the number of cells generated.

Once the cell generation routine has been entered, cell generation involves the APU retrieving a Channel Parameter Entry (from the internal VCR or externally), using the DMA Address to initiate a Memory Read operation to retrieve the SAR-SDU (size dependent on AAL type and on Gather algorithm employed), retrieving the ATM Header from the Channel Parameter Entry, modifying certain fields (GFC, PTI, CLP) if necessary, and writing the Header into the appropriate location in the VCR (just in front of where the DMA Controller was instructed to write the SAR-SDU). If the cell is an AAL 3/4 cell, the APU must also retrieve the previous SAR Header and use it the previous sequence number to generate the current SAR Header. The APU must also set the LO Field in the VCR by writing it to the end of where the DMA Controller was instructed to write the SAR-SDU after the SAR-SDU retrieval has completed (since the DMAC does not clip VCR or memory write operations on the tail end of the last word). Finally, the APU must queue the cell for transmission by writing its VCR address into the Cell Address FIFO in the ACI Transmitter. AAL 5 cells do

not require SAR Header or Trailer generation operations but they do require CRC32 Partial Results maintenance and CRC32 insertion into the last cell of a CS-PDU.

Figure 12.8 shows the Cell Generation Data Path.

Figure 12.8
Cell Generation
Data Path



Note:

1. CS-PDUs undergoing segmentation reside in system memory. SAR-SDUs are retrieved from memory (AAL 5 SAR-SDU = 48 bytes, AAL 1 SAR-SDU = 47 bytes) and placed in a Tx (Transfer) Cell Buffer in the VCR. SAR and ATM Headers are appended by the APU and then the cell is queued for transmission over the ATM Cell Interface. An eight-byte buffer (Tx Buffer) in the ACI sits between the VCR and the line driver. Data is fetched from the VCR Tx Cell Buffer relative to the ATMizer Architecture System Clock (CLK) but transferred out of the eight-byte buffer (Tx Buffer) relative to the line's byte clock (ACI's TX_CLK).

2. CS-PDUs undergoing reassembly also reside in system memory. Data from the Receiver is temporarily buffered in a second eight-byte buffer (Rc Buffer). This buffered data is then transferred to the Received Cell Buffers in the VCR. The combination of buffering provides all of the buffering needed in many applications.

CS-PDU Reassembly Process

In addition to segmentation, the ATMizer Architecture performs reassembly operations on AAL 3/4 and 5 CS-PDUs and AAL 1 realtime datastreams. In the case of AAL 5 CS-PDUs, reassembly is the process of reconstructing CS-PDUs in system memory from a stream of cells received over the ATMizer Architecture ATM Port Cell Interface. This stream of cells contains SAR-PDUs from a number of VCs/CS-PDUs simultaneously so the ATMizer Architecture has to track operations on a number of active CS-PDUs.

The exact number of open VCs that the ATMizer Architecture can support is implementation dependent. By restricting the number of active channels and caching all channel parameters in the VCR, low-cost network interface cards (NICs) can be built that use system memory for CS-PDU storage, alleviating the need for dedicated memory on the NICs themselves. In higher-speed applications, a larger number of channels (up to 65536) can be supported through the provision of external local DRAM and/or SRAM. In these implementations, the ATMizer Architecture obtains the Channel Parameter Entries from external memory.

Of course, not all high-speed (155 mbps) applications will support very large numbers of VCs. For example, an implementation of an ATM backbone may choose to encapsulate all traffic from a single network under a single VC/VP. At the destination ATM switching point, the Convergence Sublayer strips out the ATM encapsulation information exposing a diverse stream of higher-layer packets. In such systems, these high-speed ATM interface devices may wish to support only a limited number of network segments/VCs (64 – 128), and as a result, all channel parameters can be cached inside the ATMizer Architecture so local memory could consist of only DRAM.

The addition of internal memory allows ATMizer Architecture users to make several trade-offs between system cost (with or without local memory, with DRAM or with SRAM), ATM data rates, and the number of channels supported. If the external memory is capable of supplying data to the ATMizer Architecture on the clock cycle following the address, then the penalty of having an external CPE is minimal. With the four-word cache, four words can be fetched in seven cycles by strobing the first word into the APU while the other three are being fetched.

Scatter Function

When the first cell of a CS-PDU arrives over the ATMizer Architecture ATM Port Cell Interface, a buffer must be set aside in memory for reassembly. Because the ATMizer Architecture is capable of scatter operations and buffer management, it is possible to allocate buffer space one block at a time. The ATMizer Architecture can then construct a link list of the buffers used during the reassembly process, requesting additional buffer allocations as the CS-PDU extends beyond the bounds of the existing buffers.

With AAL 3/4 CS-PDUs, an intelligent decision can be made up front concerning buffer allocation since AAL 3/4 CS-PDUs contain a CS-PDU length indicator in their headers. But with AAL 5 CS-PDUs, size can not be determined until the last cell of the CS-PDU has arrived. Without a scatter capability, the system would be forced to allocate the maximum size buffer to each new AAL 5 CS-PDU, which could put a severe strain on memory resources if many channels are active simultaneously.

With scatter control, the granularity of buffer allocations can be as small as the designer wishes. User firmware running on the ATMizer Architecture retrieves available buffer lists, constructs link lists for the CS-PDUs during reassembly, and passes these lists or pointers to these lists to the Host Processor upon completion of the reassembly process (or perhaps it could append the pointer to the next buffer to the present buffer).

The ATMizer Architecture provides the resources to implement the scatter function–the APU, the DMA Controller, and ATMizer Architecture-to-Host Messaging. User firmware, downloaded to the ATMizer Architecture at system reset time, implements the buffer allocation and link list pointer management processes chosen by the system designers as the best mechanism for their application.

AAL 3/4 CRC10 Error and AAL 5 CRC32 Error Checking

If AAL 3/4 cells are to be supported, the reassembly routine has to check the CRC10 Error Bit in the System Control Register for an indication of CRC10 errors. Of course, if the Channel Parameter Entry for a VC indicates that the cell is encoded using AAL 1 or 5, CRC10 error checking should not be employed.

AAL 5 CRC32 error checking is explained in detail in Section 8.5, “CRC32 Generation.”

12.5 Congestion Notification and Handling

Switching nodes within an ATM network that is experiencing congestion can inform ATM end stations by modifying the ATM headers of ATM cells passing through them. An end station receiving marked cells may take corrective action. During reassembly, the APU can search each cell header for notification of congestion.

If congestion is indicated, the APU can execute whatever congestion handling algorithm the system designer has chosen to implement. There are several steps that the ATMizer Architecture APU can take in reaction to congestion notification:

1. The APU can inform Host software of the congestion problem but take no additional action. Host software can lower Average and Peak segmentation rates or Burst Lengths for one or more CS-PDUs/VCs.
2. The APU can react by increasing the service intervals for one or more Channel Groups (increase the initialization values in one or more PRPCs).
3. The APU can lower the Global Pacing Rate for the overall Transmission pipe.
4. The APU can choose to selectively lower the CLP value for one or more VCs. For realtime sensitive datastreams, CLP reduction may be preferable to throttling the VC.

These and other actions can be taken separately or together to achieve an efficient congestion handling mechanism. It is important to note that the hardware does not implement any congestion control algorithm. Software running on the ATMizer Architecture must check for congestion and implement a congestion control routine.

The best congestion control algorithms may not be fully understood until enough equipment is fielded to put real life demands on ATM-based networks. Much of existing equipment may not be able to be updated to deal with actual congestion problems. Systems employing the ATMizer Architecture do not have this problem. Because it is programmable, the ATMizer Architecture can execute virtually any congestion control algorithm. Because its firmware is downloaded at system reset time, software patches can be sent out to existing customer sites with new congestion algorithms for the ATMizer Architecture when more is learned about actual network congestion. Because the APU sits directly at the line interface, the ATMizer Architecture can react quickly, within a single cell time, to

congestion notification found on an incoming cell. And because it has access to the Peak Pacing Rate registers, Maximum Burst Length values, Global Pacing Rate Register and the CLP fields in the ATM headers, the ATMizer Architecture has flexibility in implementing congestion control.

12.6 Initializing the Internal Registers

The following code gives an example of how to access the ATMizer Architecture Internal Registers and enable the coprocessors internal to the APU.

```
#include "regdef.h"

#define cr_reg      r1 /* PRPC Credit Register (R) */
#define conf_reg    r2 /* PRPC Configuration Register (W) */
#define stall_reg   r3 /* PRPC Stall Register (W) */
#define cnt_init    r4 /* PRPC Count Init Register (R/W) */
#define rcv_cell    r5 /* Received Cell Pointer (R) */
#define txadr_reg   r6 /* Transmit Address FIFO (W) */
#define rcind       r7 /* Received Cell Indicator (W) */
#define gprr        r8 /* Global Pacing Rate Register (W) */
#define sys_ct_reg  r9 /* System Control Register (R/W) */
#define pr_reg      r10 /* CRC32 Register (R/W) */
#define dma_msb_reg r11 /* DMA MSB Substitution Reg (W) */
#define sp_addr     r12 /* SP Scratch Pad (R/W) */
#define hostint     r13 /* Host/DMA Port Int Addr Reg (R) */

.text
.set noreorder
.set noat

la sp_addr, 0xa0f00000     # SP Scratch Pad Address
                           # (Physical Address 0x300000)
la cr_reg, 0xfff04000      # PRU Credit Register
la conf_reg, 0xfff04100    # PRU Configuration Register
la stall_reg, 0xfff04200   # PRU Stall Register
la cnt_init, 0xfff04300    # PRPC Count Initialization Base
                           # Register
la rcv_cell, 0xfff04400    # Received Cell Pointer
la sys_ct_reg, 0xfff04a00  # System Control Register
la pr_reg, 0xfff04c00      # CRC32 Register
la hostint, 0xfff04b00     # Host/DMA Port Interrupt Register
la dma_msb_reg, 0xfff04d00 # MSB Substitution Register
```

```
/*********************************************************
Enable coprocessors so CpCond bits can be tested
*********************************************************/

li r19, 0xf0400000  # Enable all coprocessors and set
                    # bootstrap exception vector to 1
mtc0   r19, $12     # Write to cp0 Status Register

/*********************************************************
The following sequence of instructions clears the PRU CpCond
and Interrupt timeout mechanism
*********************************************************/

li r19, 0xffffffff
sw r19, (stall_reg) # Disable all counters
sw r0, (conf_reg)   # All counters are configured for
                    # CpCond2 timeout method and are
                    # clocked using the System Clock
lw r19, (cr_reg)    # Clear credit register bits associated
                    # with CpCond2 timeout method by reading
lw r19, 0x80(cr_reg)# Clear Credit Register bits associated
                    # with Interrupt1 timeout method by
                    # reading nop
sw r0, (stall_reg)  # Enable counters

/*********************************************************
The following instructions writes the values 0x30, 0x40, 0x50,
0x60, 0x70, 0x80, 0x90, 0xa0, 0xb0 and 0xc0 into counters 0,
1, 2, 3, 4, 5, 6, 7, 8, and 9 respectively. The values are
written in immediate mode (the value is written in the
initialization register as well as the PRPC itself). After the
write, the code reads from each counter.
*********************************************************/

li r14, 0x30
li r15, 0x40
li r16, 0x50
li r17, 0x60
li r18, 0x70
li r19, 0x80
li r20, 0x90
```

```
li r21, 0xa0
li r22, 0xb0
li r23, 0xc0

sh r14, (cnt_init)        # Write to counter 0
sh r15, 0x2(cnt_init)     # Write to counter 1
sh r16, 0x4(cnt_init)     # Write to counter 2
sh r17, 0x6(cnt_init)     # Write to counter 3
sh r18, 0x8(cnt_init)     # Write to counter 4
sh r19, 0xa(cnt_init)     # Write to counter 5
sh r20, 0xc(cnt_init)     # Write to counter 6
sh r21, 0xe(cnt_init)     # Write to counter 7
sw r22, 0x20(cnt_init)    # Write to counter 8
sw r23, 0x24(cnt_init)    # Write to counter 9

lh r14, (cnt_init)        # Read from counter 0
lh r15, 0x2(cnt_init)     # Read from counter 1
lh r16, 0x4(cnt_init)     # Read from counter 2
lh r17, 0x6(cnt_init)     # Read from counter 3
lh r18, 0x8(cnt_init)     # Read from counter 4
lh r19, 0xa(cnt_init)     # Read from counter 5
lh r20, 0xc(cnt_init)     # Read from counter 6
lh r21, 0xe(cnt_init)     # Read from counter 7
lw r22, 0x20(cnt_init)    # Read from counter 8
lw r23, 0x24(cnt_init)    # Read from counter 9

/*********************************************************
The following instruction illustrates a load from the Host/DMA
Port memory space into the APU Register R20 and then a store
from the APU Register R20 into the VCR. The ATMizer
Architecture is big-endian on both ports as well as
internally.
*********************************************************/

li r19, 0x282             # 10 MSB of Main Memory Address
sw r19, (dma_msb_reg)     #
la r19, 0x9d000c          # Main Memory Effective Address
lw r20, (r19)
la r21, 0xfff00fcc        # VCR Address
sw r20, (r21)             # Store into the VCR
```

```
        /*********************************************************
        The following instruction causes the ATMizer Architecture to
        assert HBS_INT
        *********************************************************/

        sw r0, (hostint)          # Generate interrupt

        /*********************************************************
        The following loop waits for receive cell
        *********************************************************/

loop:   bc1f loop     # Check for CpCond1, if not asserted, loop here
        nop
```

Chapter 13
Functional Waveforms

This chapter contains and describes the ATMizer Architecture functional waveforms.

This chapter has five sections:

- Section 13.1, “Secondary Port”
- Section 13.2, “Host/DMA Port”
- Section 13.3, “Serial Interface”
- Section 13.4, “ACI Transmitter”
- Section 13.5, “ACI Receiver”

Note In some places within the waveforms, the letter A represents Address, the letter B represents Byte, the letter D represents Data, and the letter W represents Word.

13.1 Secondary Port

Figure 13.1 shows the ATMizer Architecture performing a single-word read and a single-word write through the Secondary Port. First the ATMizer Architecture asserts SP_RQ to initiate a transaction. The external arbiter gives the bus grant to the ATMizer Architecture by asserting SP_GNT. The ATMizer Architecture generates valid addresses on SP_AD[31:0] after external logic asserts SP_GNT. External logic latches the address and control information at the rising edge of CLK when SP_GNT and SP_ASEL are both asserted HIGH. After external logic has latched the address, it deasserts SP_ASEL to start the data phase of the transaction.

For a read operation, the ATMizer Architecture detects $\overline{\text{SP\_ACK}}$ LOW at the rising edge of CLK and latches the data on rising edge of CLK one clock cycle later.

For a write operation, the ATMizer Architecture drives write data after external logic deasserts SP_ASEL LOW. One clock cycle after it detects $\overline{\text{SP\_ACK}}$ LOW, the ATMizer Architecture terminates the transaction by deasserting SP_RQ.

Figure 13.1 Secondary Port One-Word Read and Write

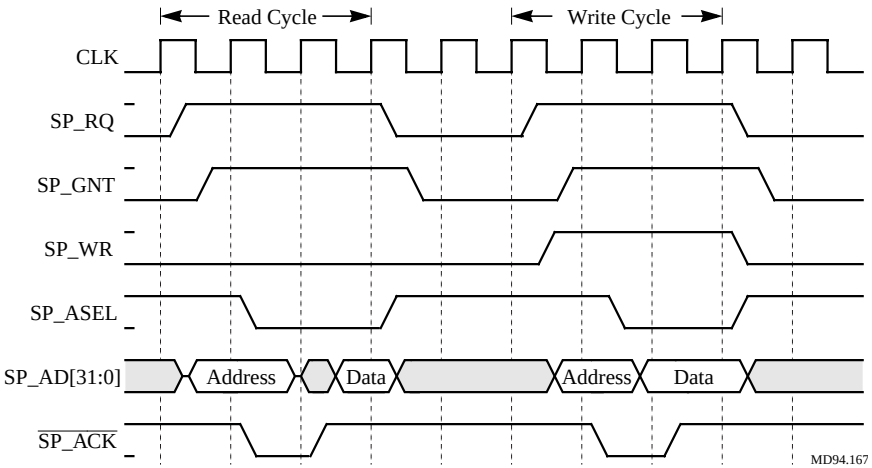


Figure 13.2 shows the fastest access through the Secondary Port. SP_GNT is tied HIGH all the time, so the ATMizer Architecture presents the address on SP_AD[21:0] one cycle before asserting SP_RQ. This technique allows external logic to latch the address one clock cycle earlier. Whenever the ATMizer Architecture asserts SP_RQ, external logic can assert $\overline{\text{SP\_ACK}}$. The ATMizer Architecture can terminate the transaction on the next clock cycle, resulting in a two-clock-cycle access. Unlike SP_AD[21:0], SP_AD[31:28], the Write Byte Enables, are valid at the same cycle that the ATMizer Architecture asserts SP_RQ.

Figure 13.2
Secondary Port
Fastest Access

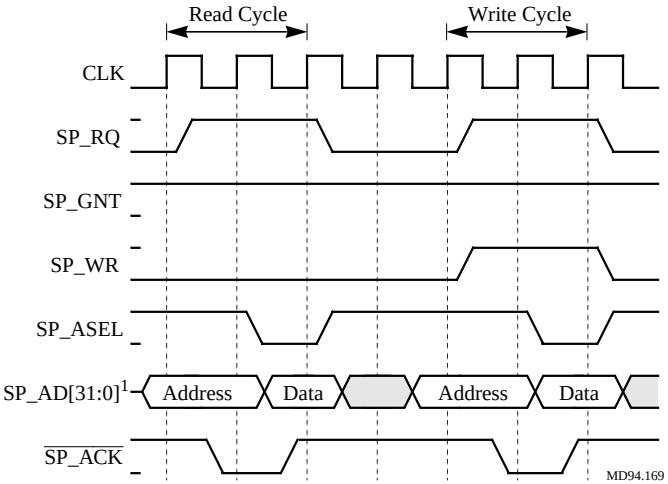


Figure 13.3 shows the ATMizer Architecture performing a block fetch operation through the Secondary Port. As in Figure 13.1, the ATMizer Architecture starts the transaction by asserting SP_RQ. External logic asserts SP_GNT, and the ATMizer Architecture generates the address on SP_AD[31:0]. External logic latches the address and control information at the rising edge of CLK when SP_GNT and SP_ASEL are both asserted HIGH, and asserts $\overline{\text{SP\_ACK}}$ for four clock cycles starting from the next CLK pulse. SP_AD25 should be latched as the Block Fetch Bit. The ATMizer Architecture latches the data on the next four CLK pulses. In this example, external address latch wraps around the address on two-word or four-word boundaries depending on the Block Fetch size. After the ATMizer Architecture latches the last word of data, it terminates the transaction by deasserting SP_RQ.

Figure 13.3
Secondary Port
Four-Word Read

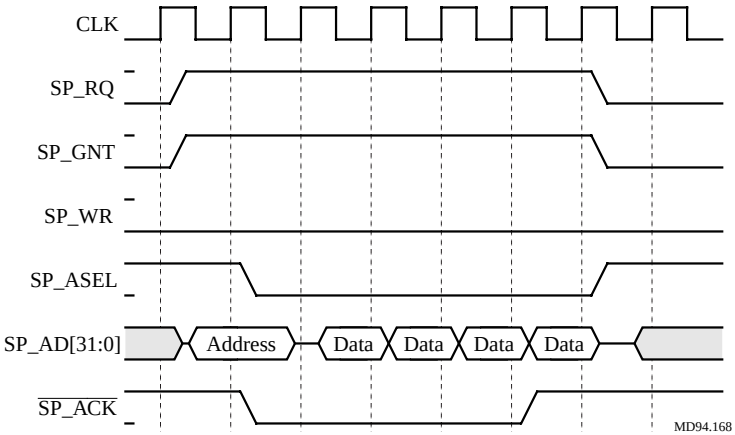


Figure 13.4 shows the same function as Figure 13.3, except in this example, external logic toggles SP_ASEL during the word operations. If the external address latch cannot increment the address based on a four-word boundary as in Figure 13.3, external logic can assert SP_ASEL HIGH for one or more cycles to make the ATMizer Architecture drive the next address. The external address latch can then latch the next address.

Figure 13.4
Secondary Port Four-
Word Read with SP_ASEL
Toggle

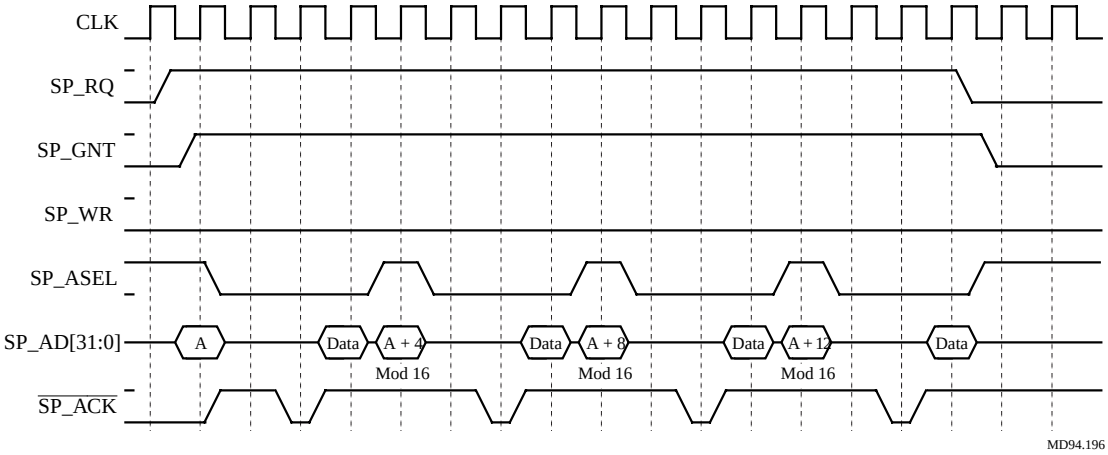
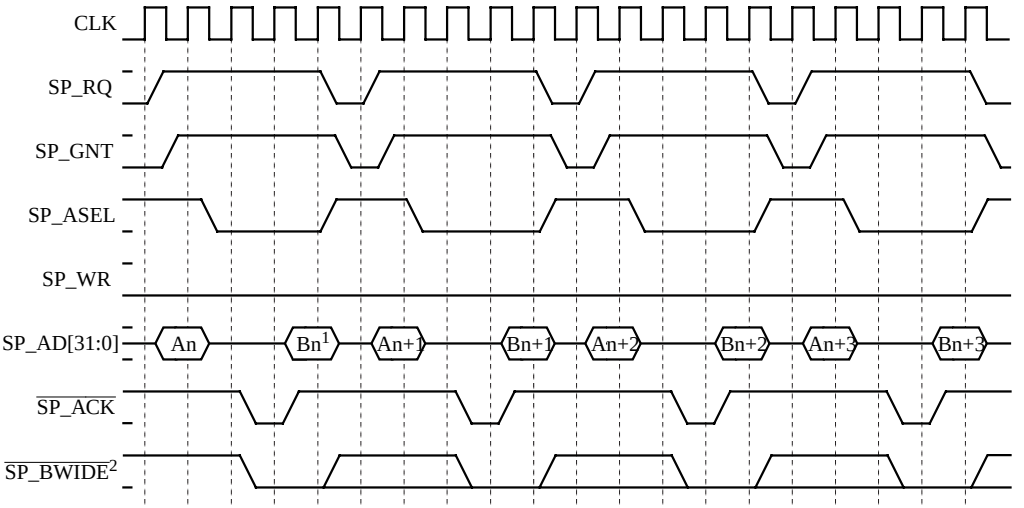


Figure 13.5 shows the ATMizer Architecture communicating with a byte-wide device, such as a Boot PROM. The byte-wide device's D[7:0] must

be connected to SP_AD[7:0]. When the ATMizer Architecture starts the access by presenting an address on SP_AD[31:0], external logic must assert $\overline{\text{SP\_BWIDE}}$ and $\overline{\text{SP\_ACK}}$ to inform the ATMizer Architecture that it is a byte-wide device. The ATMizer Architecture then performs three subsequent accesses, incrementing the address by one each time. External logic must assert $\overline{\text{SP\_BWIDE}}$ during all four accesses.

Figure 13.5
Dynamic Bus Sizing on Secondary Port



Note:
1. During a byte access, the ATMizer Architecture always expects byte data on SP_AD[7:0].

13.2 Host/DMA Port

Figure 13.6 shows the ATMizer Architecture performing a single-word load/store operation through the Host/DMA Port. The ATMizer Architecture initiates an access by asserting HBS_RQ. The external arbiter grants the bus to the ATMizer Architecture by asserting HBS_GNT. The following clock cycle, the ATMizer Architecture puts the correct address onto HBS_A[31:2], sets the correct byte enables on $\overline{\text{HBS\_BE}}$[3:0], and asserts $\overline{\text{HBS\_AS}}$, the address strobe.

HBS_WR deasserted indicates a read operation. HBS_WR asserted indicates a write operation. For a read operation, whenever external logic asserts $\overline{\text{HBS\_ACK}}$, the ATMizer Architecture latches the data on HBS_D[31:0] and terminates the transaction by deasserting HBS_RQ. For

a write operation, the ATMizer Architecture sources the address on HBS_A[31:2] and data on HBS_D[31:0] at the same time.

For a single-word access as shown in Figure 13.6, the ATMizer Architecture asserts $\overline{\text{HBS\_END}}$ to inform external logic that it is waiting for the last acknowledgment, $\overline{\text{HBS\_ACK}}$.

When the ATMizer Architecture detects the deassertion of $\overline{\text{HBS\_ACK}}$, it terminates the operation.

Figure 13.6 Direct Load/Store Word Through DMA Port

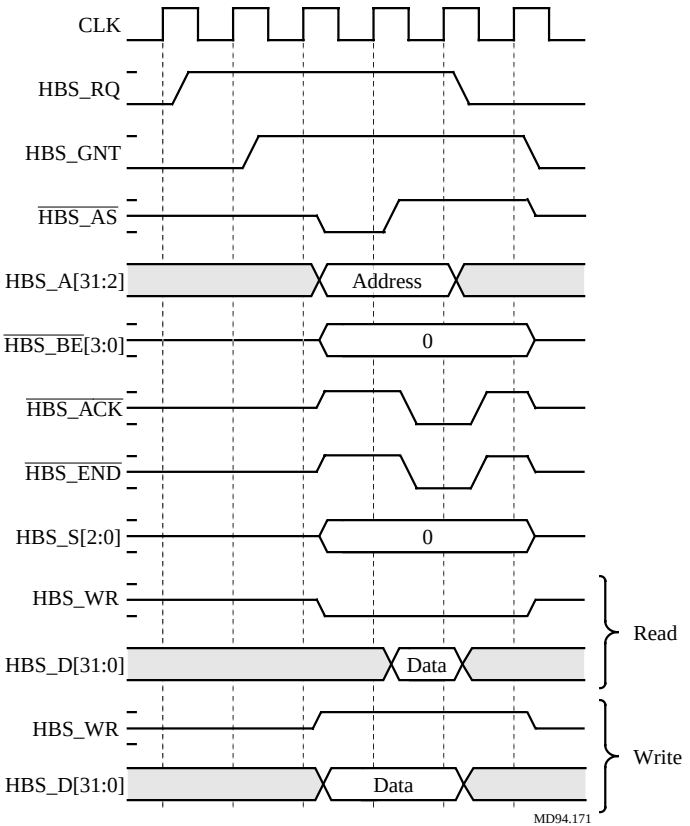


Figure 13.7 shows a 16-byte DMA operation. The ATMizer Architecture starts the DMA operation by asserting HBS_RQ (CLK Pulse 1). External logic grants the bus by asserting HBS_GNT one clock cycle later (CLK Pulse 2). On the next clock cycle (CLK Pulse 3), the ATMizer Architecture sources the address on HBS_A[31:2], the byte enables on $\overline{\text{HBS\_BE}}$[3:0], and the 16-byte DMA SIZE signals for SBus operation on HBS_S[2:0]. The ATMizer Architecture also asserts $\overline{\text{HBS\_AS}}$ and deasserts

HBS_END. The ATMizer Architecture detects the first HBS_ACK assertion by external logic, and sources the next address and data (CLK Pulse 5). The ATMizer Architecture does not source the next address and data until it detects another HBS_ACK assertion. In this example, HBS_ACK remains asserted throughout the transaction, so the next addresses and data are immediately sourced. At the rising edge of CLK Pulse 7, since the ATMizer Architecture has sourced 12 bytes, it asserts HBS_END to inform the external device that the current transfer will end the next time external logic asserts HBS_ACK. At the rising edge of CLK Pulse 8, the ATMizer Architecture detects the last HBS_ACK assertion, and terminates the DMA operation.

Figure 13.7 16-Byte DMA Transactions

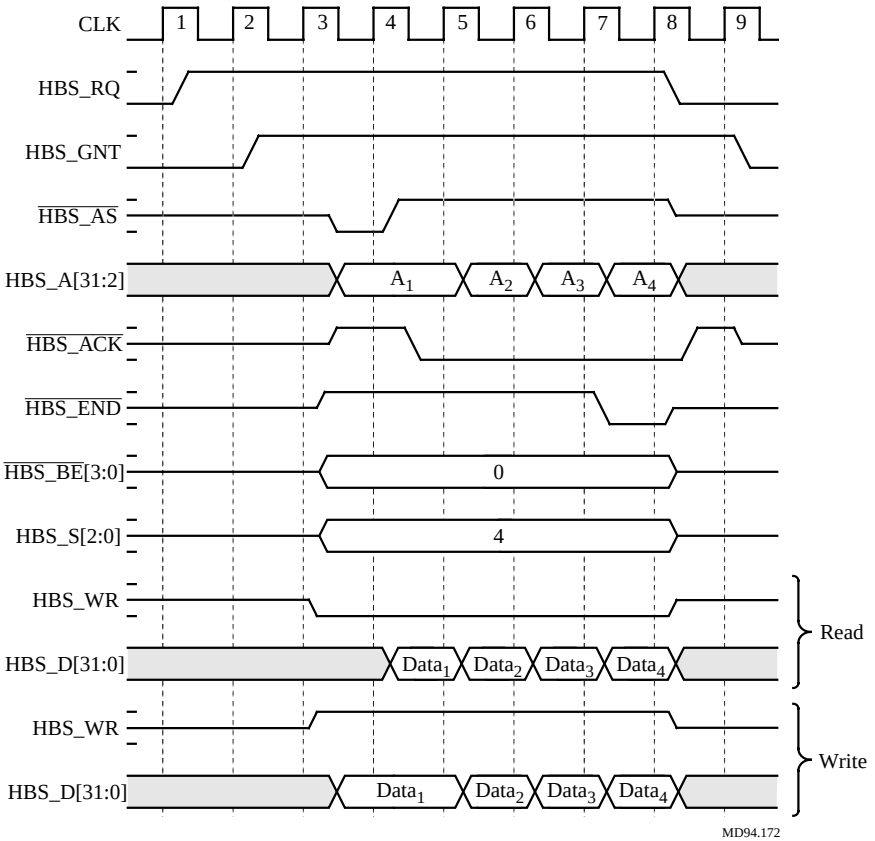


Figure 13.8 shows how the DMA Engine works when a DMA operation does not align on a word boundary. This example uses a 16-byte DMA operation with an address offset of 0x2. The access starts at address A_1

with $\overline{\text{HBS\_BE}}$[3:0] = 0xC (a read of the lower two bytes), followed by three four-byte DMA read operations at addresses A_2, A_3, and A_4. The fifth operation should be a halfword (two-byte) access with $\overline{\text{HBS\_BE}}$[3:0] equal to 0x3. Since the last word of DMA operation of the ATMizer Architecture has to be a word access, the ATMizer Architecture performs the fifth access as a word operation, resulting in an 18-byte DMA operation.

Figure 13.8 Non-Word-Aligned 16-Byte DMA Transactions

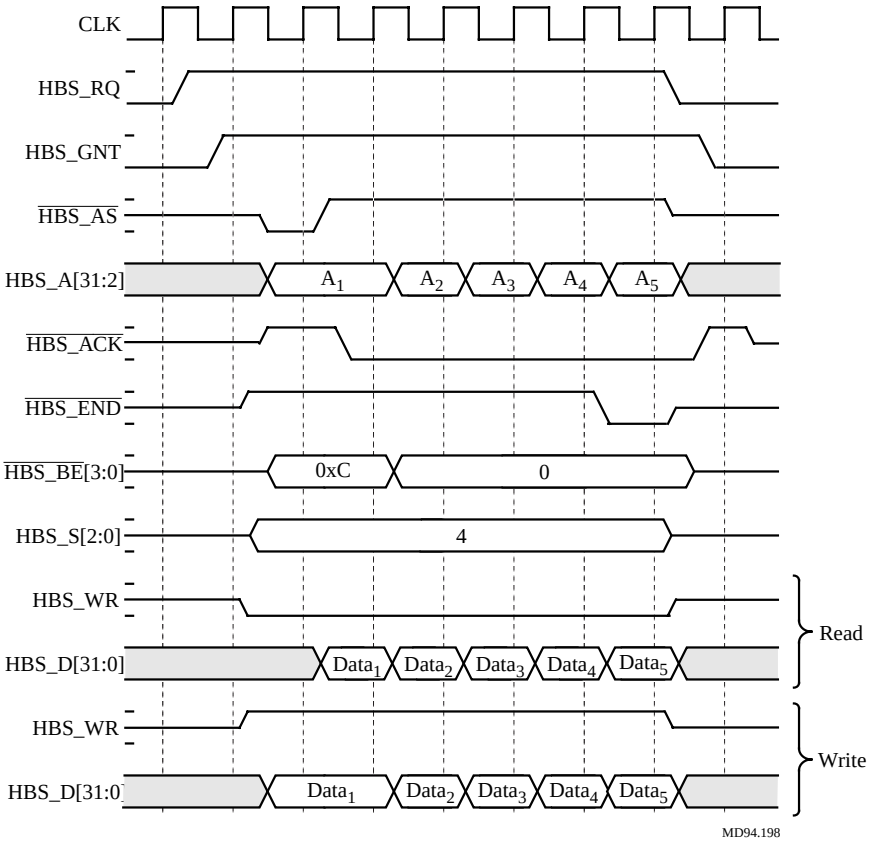


Figure 13.9 shows the ATMizer Architecture performing an atomic operation through the Host/DMA Port. The first read accesses a four-word block. When the ATMizer Architecture finishes the first block fetch operation, it keeps HBS_RQ asserted, and a second operation (a write) occurs later.

Figure 13.9
Block Fetch followed by a
Single-Word Store

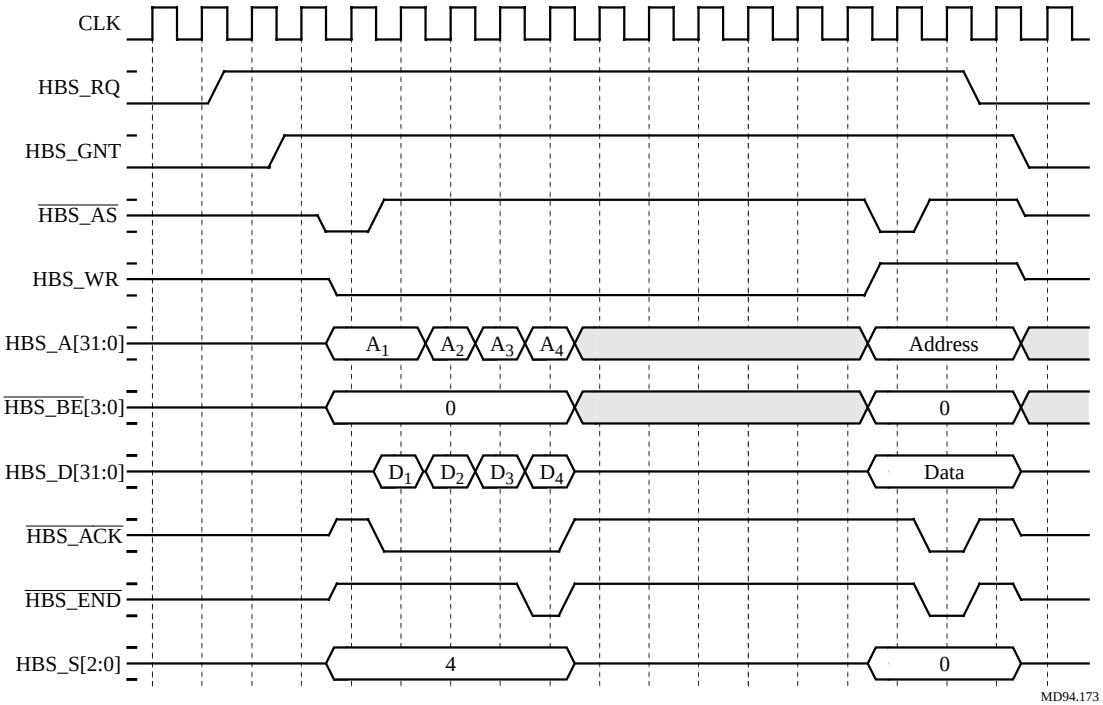


Figure 13.10 shows a 16-byte DMA read with a CPU steal cycle. The ATMizer Architecture starts the DMA operation and detects the assertion of $\overline{\text{HBS\_ACK}}$. Internally, the APU (CPU) issues a write operation. Since the APU has a higher priority than the DMAC, the ATMizer Architecture suspends the DMA operation by asserting $\overline{\text{HBS\_END}}$ after detecting the second $\overline{\text{HBS\_ACK}}$ assertion. The ATMizer Architecture reasserts $\overline{\text{HBS\_AS}}$ along with outputting a new CPU write address and data. After the ATMizer Architecture detects the $\overline{\text{HBS\_ACK}}$ assertion for the CPU-steal cycle, it reasserts $\overline{\text{HBS\_AS}}$ and resumes the stalled DMA operation.

Figure 13.10
16-Byte DMA Transaction
with CPU Steal Cycle

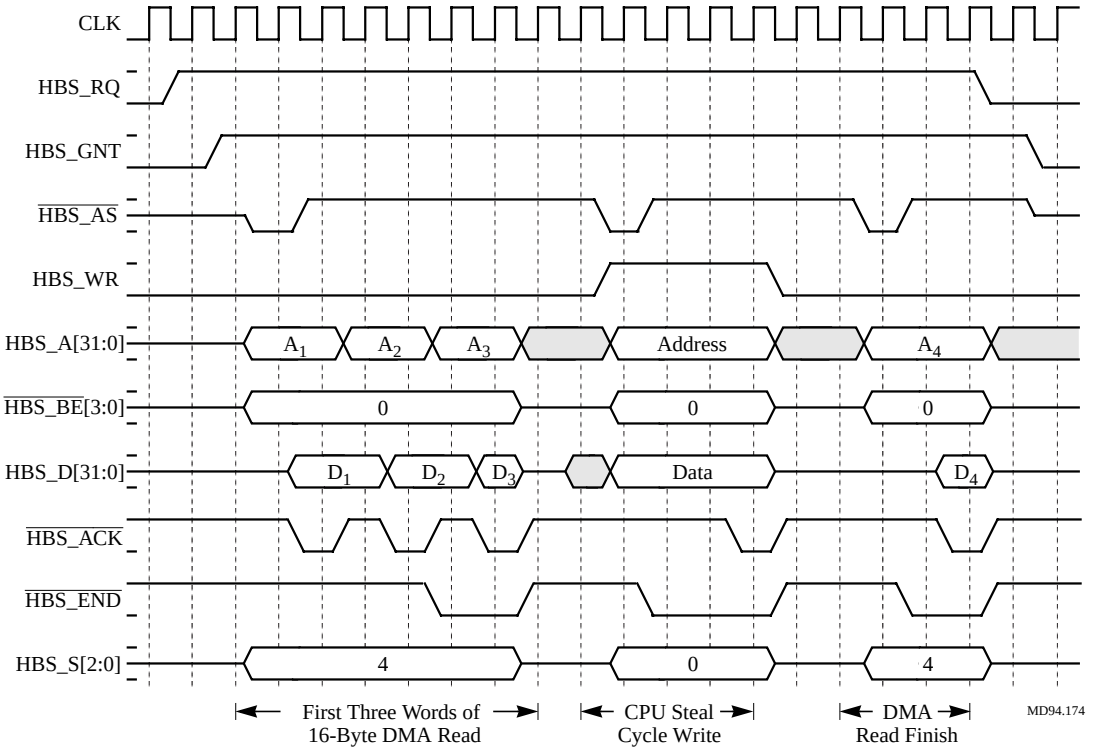


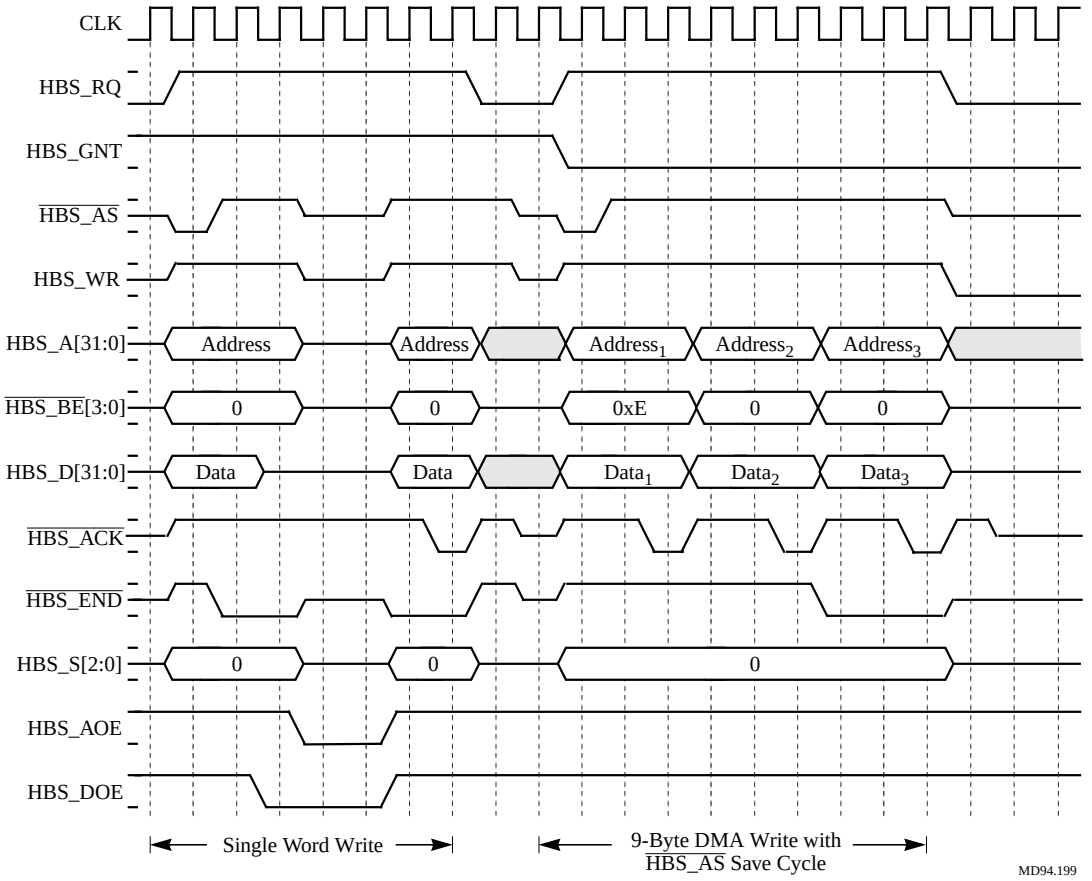
Figure 13.11 shows Host/DMA Port operation with HBS_AOE, HBS_DOE Toggle and $\overline{\text{HBS\_AS}}$ Save Cycle.

Figure 13.11 shows how the Host/DMA Port operates with HBS_GNT tied HIGH all the time, and how HBS_AOE and HBS_DOE affect the Host/DMA Port output pins. The first operation is a single-word write operation. In this case, the ATMizer Architecture asserts $\overline{\text{HBS\_AS}}$ at the same cycle it asserts HBS_RQ, which saves one bus clock cycle. The single-word write operation also shows HBS_AOE and HBS_DOE deasserted for two to three cycles, which makes the Host/DMA Port 3-state all of the control outputs ($\overline{\text{HBS\_AS}}$, HBS_WR, $\overline{\text{HBS\_BE}}$[3:0], $\overline{\text{HBS\_END}}$, HBS_S[2:0]), the address bus (HBS_A[31:2]), and the data bus (HBS_D[31:0]).

The following operation is a nine-byte DMA write where the external arbiter deasserts HBS_GNT at the same cycle the ATMizer Architecture asserts HBS_RQ. If the ATMizer Architecture detects HBS_GNT asserted

at the rising edge of CLK when it asserts HBS_RQ, the ATMizer Architecture initiates the transaction even though the external arbiter deasserts HBS_GNT one cycle later. The nine-byte DMA operation shows that the ATMizer Architecture does not have the bus grant throughout the entire bus cycle, but it still performs all operations.

Figure 13.11
Host/DMA Port Operation with HBS_AOE, HBS_DOE Toggle and $\overline{\text{HBS\_AS}}$ Save Cycle



13.3 Serial Interface

Figure 13.12 shows a serial downloading of x words. When external logic deasserts $\overline{\text{RST}}$, the ATMizer Architecture checks the status of $\overline{\text{SRL\_BOOT}}$. If $\overline{\text{SRL\_BOOT}}$ is asserted LOW, the ATMizer Architecture enables the serial downloading process and generates SRL_CLK16 (system clock divided by 16), which can drive a serial device such as a Serial PROM. Sometime after external logic deasserts $\overline{\text{RST}}$, the serial device drives Bit 31 Word 0 onto SRL_DIN and asserts SRL_ACK. If SRL_ACK is asserted, the ATMizer Architecture latches in one bit of data on every rising edge of SRL_CLK16. If the serial device cannot provide data every SRL_CLK16 pulse, it can deassert SRL_ACK to inform the ATMizer Architecture not to latch bit data for the following rising edge of SRL_CLK16. When the external serial device is able to provide data again, it asserts SRL_ACK back to HIGH, and at the next rising edge of SRL_CLK16, the ATMizer Architecture latches the following bit. External logic deasserts $\overline{\text{SRL\_BOOT}}$ to HIGH after the ATMizer Architecture has latched Bit 0 Word (X-1). Then the ATMizer Architecture starts fetching instructions either from the Host/DMA Port or the Secondary Port.

Figure 13.13 shows the completion of a 4-Kword serial download. The maximum downloading size for each downloading is 4 Kwords. The waveform shows an external serial device providing bit data with no wait state, so SRL_ACK is asserted HIGH all the time. After the ATMizer Architecture has latched the last bit (Bit 0 of Word 4095), the external device deasserts $\overline{\text{SRL\_BOOT}}$ to HIGH to cause the ATMizer Architecture to boot from either the Host/DMA Port or Secondary Port.

Figure 13.14 shows two successive serial downloads, used when the user firmware requires more than 4 Kwords. The first download is the same as that in Figure 13.13. After the first download is complete, external logic deasserts $\overline{\text{SRL\_BOOT}}$ HIGH for at least 32 system clock cycles and then asserts it LOW again to start the second download. SRL_CLK16 stays HIGH whenever $\overline{\text{SRL\_BOOT}}$ is deasserted HIGH. When $\overline{\text{SRL\_BOOT}}$ is deasserted the first time, the ATMizer Architecture starts fetching instructions from the Host/DMA or Secondary Port. After the ATMizer Architecture starts booting from the Host/DMA or Secondary Port, both APU instruction fetch and Serial downloading processes occur at the same time. If user firmware requires more than 8 Kwords, a multiple downloading process can be implemented.

Figure 13.12
Serial Downloading Less Than 4 Kwords

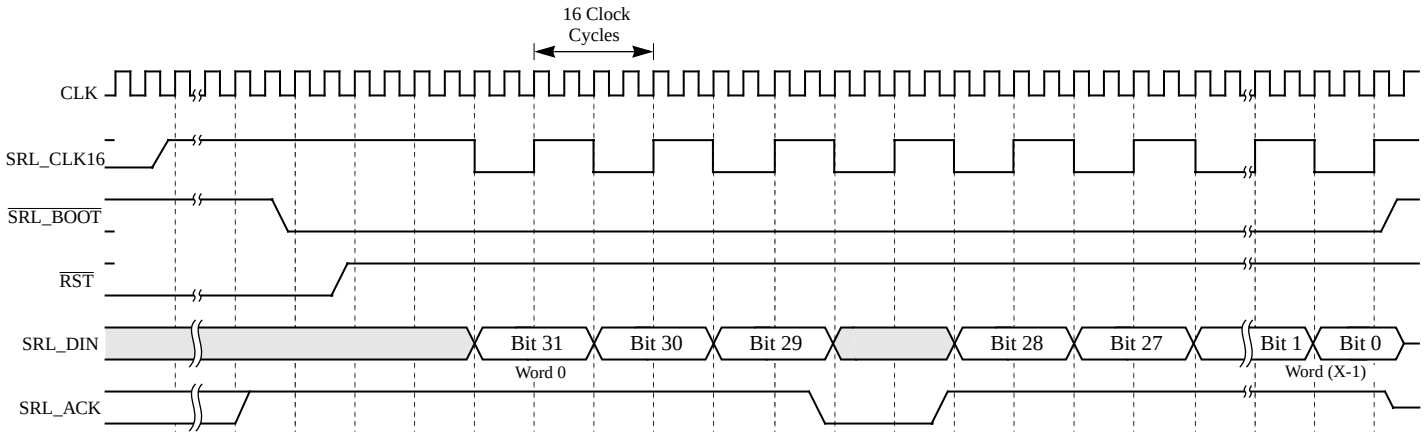


Figure 13.13
Completion of 4-Kword Serial Downloading

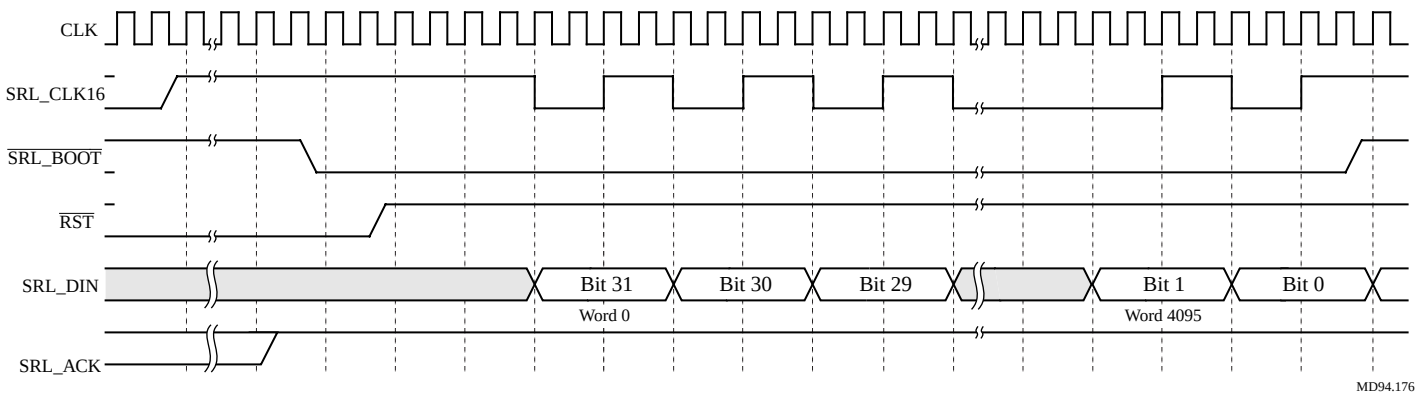
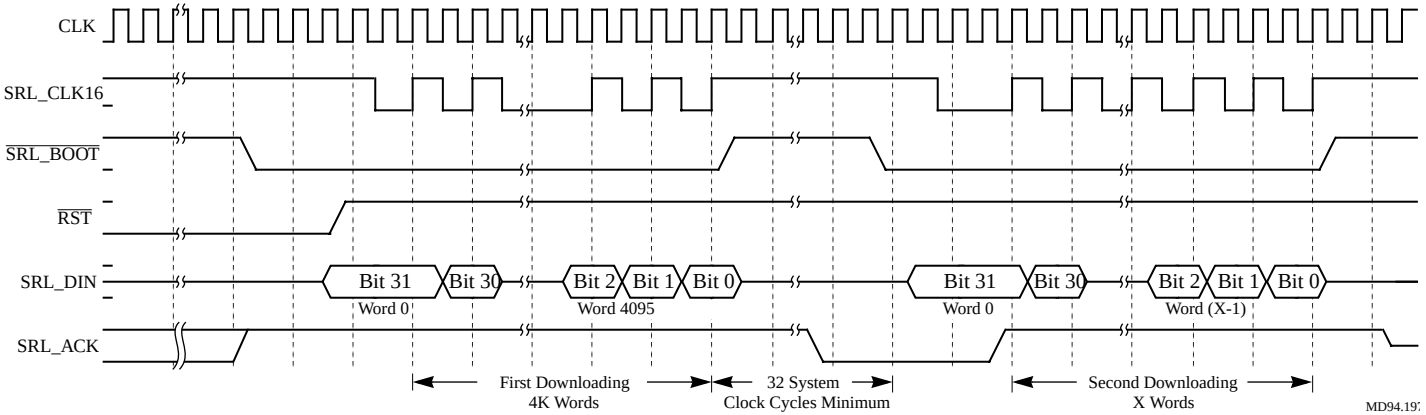


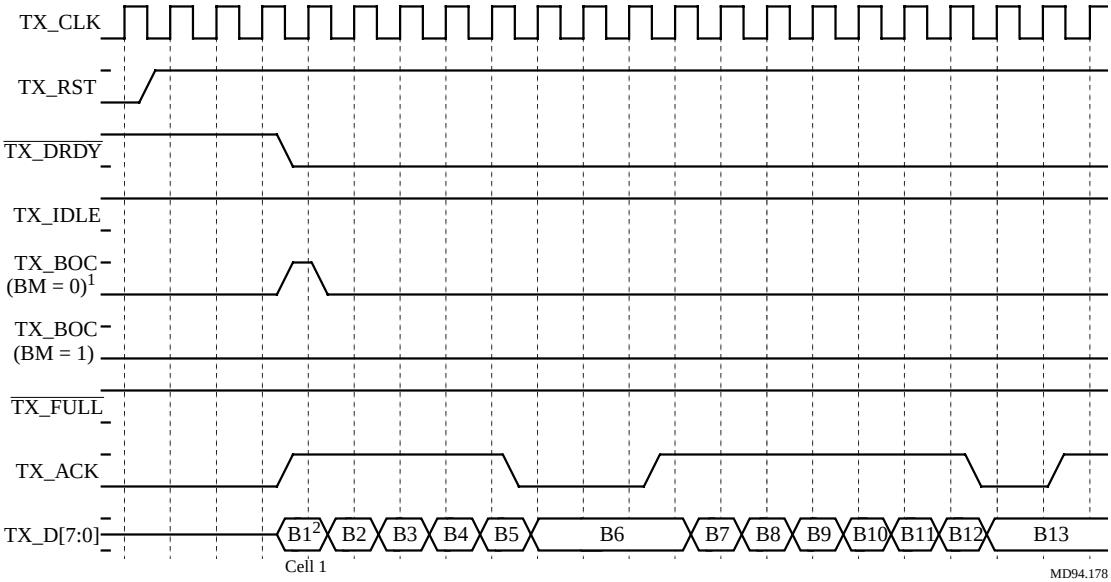
Figure 13.14
Multiple Serial Downloading



13.4 ACI Transmitter

Figure 13.15 shows the user firmware initializing the ACI Transmitter and the ATMizer Architecture transmitting the first Idle Cell. The ATMizer Architecture asserts TX_IDLE HIGH when the system reset, $\overline{\text{RST}}$, is deasserted. When user firmware sets the TI Bit in System Control Register to one, the ATMizer Architecture ACI module deasserts $\overline{\text{TX\_RST}}$ and starts to transmit. Three clock cycles after deasserting $\overline{\text{TX\_RST}}$, the ATMizer Architecture sources TX_D[7:0] and asserts $\overline{\text{TX\_DRDY}}$ and TX_BOC. External logic asserts TX_ACK when it has latched the byte data. Whenever TX_ACK is asserted on the rising edge of TX_CLK, the current byte is latched by the framing logic and new byte data is sourced on TX_D[7:0] by the ATMizer Architecture. After receiving Byte 5, the framing logic cannot keep up with the ATMizer Architecture so it deasserts TX_ACK for three TX_CLK cycles, which makes Byte 6 remain sourced on TX_D[7:0]. When external logic receives Byte 6, it asserts TX_ACK again, and the ATMizer Architecture starts sourcing new byte data on TX_D[7:0].

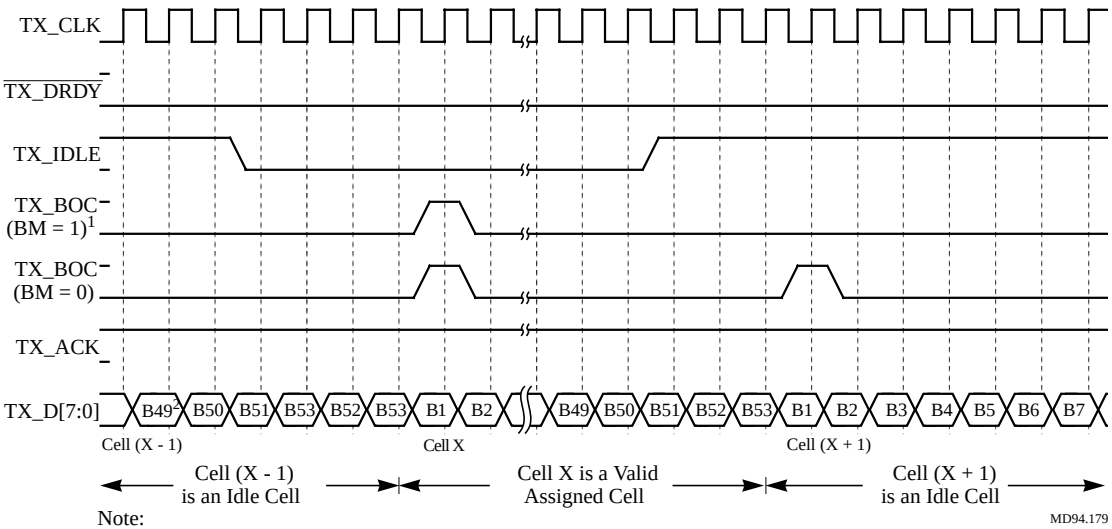
Figure 13.15
ACI Transmitter
Initialization



Note:
1. BM refers to the BM Bit in the System Control Register.
2. B1 = Byte 1 of Cell 1.

Figure 13.16 shows the ATMizer Architecture transmitting an Assigned Cell. The ATMizer Architecture asserts $\overline{\text{TX\_DRDY}}$ and transmits the first Idle Cell. When the ATMizer Architecture has an Assigned Cell to transmit, it deasserts TX_IDLE four cycles before asserting TX_BOC and sourcing the cell on TX_D[7:0]. If the cell following the Assigned Cell is an Idle Cell, the ATMizer Architecture asserts TX_IDLE three cycles before sourcing the Idle Cell on TX_D[7:0]. Setting the BM Bit in the System Control Register to one causes the ACI Transmitter to deassert TX_BOC at the beginning of Idle Cell transmission (SAI mode). Clearing BM to zero causes the ACI Transmitter to assert TX_BOC at the beginning of Idle Cell transmission (UTOPIA mode).

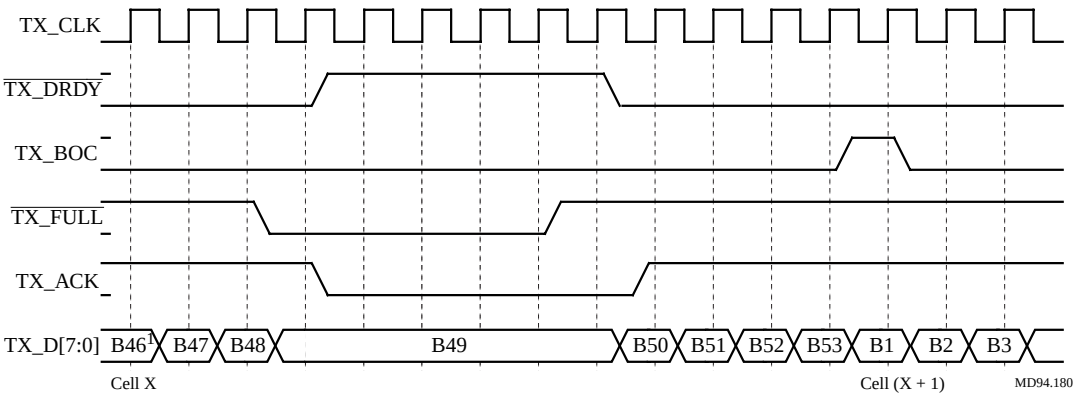
Figure 13.16
ACI Assigned Cell Transmission



Note:
1. BM refers to the BM Bit in the System Control Register.
2. B49 = Byte 49 of Cell (X-1).

Figure 13.17 shows the behavior of the ATMizer Architecture when the framing logic chip asserts $\overline{\text{TX\_FULL}}$. When the ATMizer Architecture detects $\overline{\text{TX\_FULL}}$ asserted, it deasserts $\overline{\text{TX\_DRDY}}$ on the next cycle and holds the data on TX_D[7:0]. When the framing logic chip asserts $\overline{\text{TX\_FULL}}$ back to HIGH (inactive), the ATMizer Architecture asserts $\overline{\text{TX\_DRDY}}$ on the next cycle and sources the next byte to be transmitted on TX_D[7:0].

Figure 13.17
ACI $\overline{\text{TX\_FULL}}$ Assertion

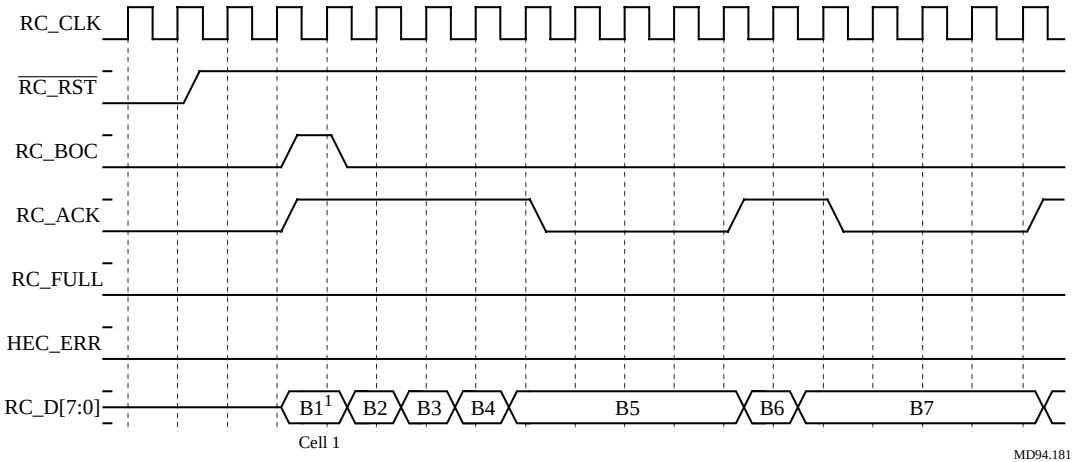


Note:
1. B46 = Byte 46 of Cell X.

13.5 ACI Receiver

Figure 13.18 shows the initialization of the ACI Receiver. After user firmware sets the RI Bit in the System Control Register, the ATMizer Architecture deasserts $\overline{\text{RC\_RST}}$ to indicate that it is ready to receive cells. When the ATMizer Architecture detects the first assertions of RC_BOC and RC_ACK, it latches the first byte of the first cell and then latches the following bytes. As the ATMizer Architecture transmits a cell, if the framing logic cannot keep up with the ATMizer Architecture, external logic can deassert RC_ACK for several cycles, causing the ATMizer Architecture to ignore the data on RC_D[7:0] until RC_ACK is reasserted again.

Figure 13.18
ACI Receiver Initialization



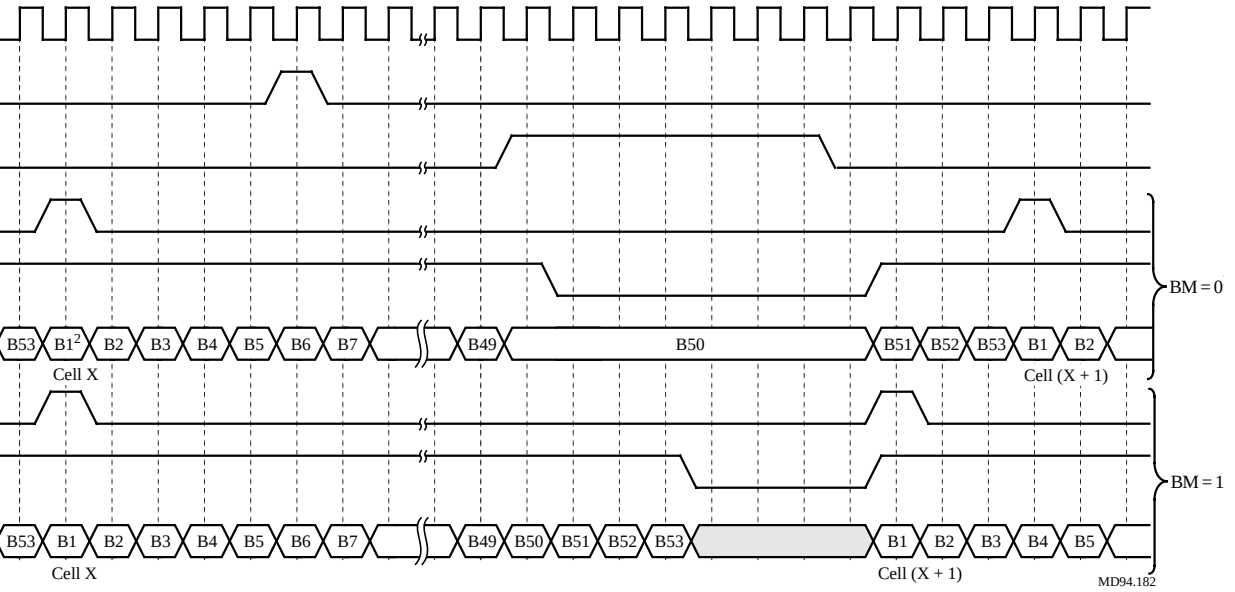
Note:
1. B1 = Byte 1 of Cell 1.

Figure 13.19 shows the ATMizer Architecture asserting RC_FULL HIGH when the internal Received Cell Buffer (Holder) in the VCR is almost full (six bytes before it is full). When the BM Bit in the System Control Register is zero (UTOPIA Mode), the ATMizer Architecture receives only one more byte on RC_D[7:0] after asserting RC_FULL. When BM Bit in the System Control Register is one (SAI Mode), the ATMizer Architecture finishes receiving the cell completely after asserting RC_FULL.

Several cycles later, after the APU has processed the cell, it frees up the VCR and the ACI Module deasserts RC_FULL. Then the ATMizer Architecture is able to continue receiving the remaining bytes (UTOPIA Mode) or to latch data at the beginning of the next cell (SAI Mode).

Also in this example, the ATMizer Architecture expects a HEC byte on the fifth byte and asserts HEC_ERR on the sixth byte because there is an error.

Figure 13.19
ACI Receiver HEC_ERR
and RC_FULL Assertion



rs to the BM Bit in the System Control Register.
te 1 of Cell X.

Chapter 14
Registers

This chapter describes all the ATMizer Architecture registers.

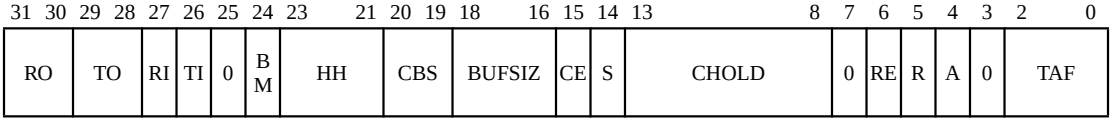
This chapter has three sections:

- Section 14.1, “System Control Register”
- Section 14.2, “APU Core Registers”
- Section 14.3, “Other Registers Summary”

14.1 System Control Register

Certain functions within the ATMizer Architecture are programmable and must be configured at system reset time. All ATMizer Architecture configuration information is stored in the System Control Register that is written by the APU as part of its initialization routine. The System Control Register is programmed using a store word instruction to Effective Address 0xFFF04A00.

Figure 14.1
System Control Register



RO **Receive Offset** **[31:30]**
This field configures the receive cell size. The ATMizer Architecture supports receive cell sizes of 52 (53 if HEC is enabled on transmit), 56 (57), 60 (61), and 64 (65). All cells are built into the VCR so that the last byte of the SAR Payload is aligned to the 64th byte of the cell buffer. The transmit and receive cell sizes do

not have to be the same. The following table shows how setting RO sets the cell size:

RO	*Cell Size (Bytes)*
00_2	64
01_2	60
10_2	56
11_2	52

TO **Transmit Offset** **[29:28]**

This field sets the transmit cell size. The ATMizer Architecture supports transmit cell sizes of 52 (53 if HEC is enabled on transmit), 56 (57), 60 (61,) and 64 (65). All cells are built into the VCR so that the last byte of the SAR Payload is aligned to the 64th byte of the cell buffer. The transmit and receive cell sizes do not have to be the same. The following table shows how setting TO sets the cell size:

TO	*Cell Size (Bytes)*
00_2	64
01_2	60
10_2	56
11_2	52

RI **Receive Initialize** **27**

This bit has a direct impact in synchronizing the ACI Receiver. Setting RI to one causes the ATMizer Architecture to deassert $\overline{\text{RC\_RST}}$. Before setting RI, firmware must initialize the ACI Receiver parameters, such as: the receiver offset, the buffer size, and HEC handling. Once $\overline{\text{RC\_RST}}$ is deasserted, the ACI Receiver can begin receiving cells. The receiver can be initialized again at any time by clearing RI to zero. For more information on how to initialize the receiver, refer to Section 9.5, “ACI Receiver.”

TI **Transmitter Initialize** **26**

Clearing TI to zero puts the ACI Transmitter in reset mode. Setting TI to one puts the ACI Transmitter out of reset mode and into normal mode. All other transmitter related parameters must be defined before setting this bit to one. The ACI Transmitter can

be initialized again during normal operation by clearing TI to zero.

BM **BOC Mode** **24**

Setting BM to one puts the ACI Transmitter into SAI Mode. Clearing BM to zero puts the ACI Transmitter into UTOPIA Mode. For more information on the differences between SAI and UTOPIA Modes, refer to Section 9.9, “Interfaces.”

HH **HEC Handling** **[23:21]**

The HEC transmission circuitry is separate from the HEC reception circuitry. Bit 21 controls HEC during cell transmission and Bits [23:22] handle HEC during cell reception.

Setting Bit 23 to one causes the ATMizer Architecture to check the validity of the HEC. If an error is encountered, the ATMizer Architecture notifies the Physical Layer by asserting HEC_ERR. Clearing Bit 23 to zero causes the ATMizer Architecture to accept HEC, but ignore it.

Setting Bit 22 to one causes the ATMizer Architecture to expect HEC in Byte 5 of the ATM Header during cell arrival. Clearing Bit 22 to zero causes the ATMizer Architecture to not expect HEC in Byte 5 of the ATM Header during cell arrival.

Setting Bit 21 to one causes the ACI Transmitter to generate and insert HEC into Byte 5 of the ATM Header. Clearing Bit 21 to zero causes the ACI Transmitter to not generate HEC.

When Bits 22 and 23 are both set to one, the ACI checks the validity of the HEC in Byte 5 of the ATM Header. If the ACI detects a HEC error, it discards the cell and asserts HEC_ERR.

HH	***HEC Mode[1]***
Bit 23	Check for HEC errors on receive
Bit 22	Expect HEC on receive
Bit 21	Generate and insert HEC on transmit

1. If bit set to one.

CBS **Cache Block Size** **[20:19]**

The Cache Block Size determines the number of words that are burst during a data cache read miss. Clearing CBS to 00_2 makes the burst size two words. Setting CBS to 01_2 makes the burst size four words. Other values are currently invalid and are reserved

for future use. Note that this value must be the same as Bits [5:4] of the APU BIU/Cache Configuration Register.

BUFSIZ **Buffer Size** **[18:16]**

Buffer Size determines the size of the Received Cell Holder Buffer. Received cells are written into the VCR, one cell per 64-byte block. Bit 18 is always zero for the current implementation and it is reserved for future use. The following table shows how setting BUFSIZ sets the buffer size:

BUFSIZ	*Buffer Size (Number of Cells)*
000_2	4
001_2	8
010_2	16
011_2	32

CE **CRC10 Error** **15**

User firmware checks the CRC10 Error Bit when it starts to reassemble an AAL 3/4 cell. The ATMizer Architecture sets this bit to one to indicate that the current cell has a CRC10 error. The ATMizer Architecture clears this bit to zero to indicate that the current cell is free of CRC10 errors, so the firmware can continue to process the cell. The APU must process cells in the order they were stored in the VCR in order to match the cell with it's corresponding CRC10 Error Bit.

If the APU is processing an AAL 5 cell, it does not have to check the CRC10 Error Bit. The CRC10 Error Bit can be checked by executing two instructions. The first instruction is a Load Halfword to load Bits [15:0] of the System Control Register into one of the general purpose registers (such as R5). The load treats Bit 15, which is the CRC10 Error Bit, as the sign bit. The second instruction is a Branch on R5 Less Than Zero. If the branch is taken, then there is a CRC10 error, otherwise the cell is free of CRC10 errors.

S **Safety** **14**

User firmware must clear this bit to zero for normal operation. During normal operation, the ATMizer Architecture resynchronizes the ACI Receiver to the proper cell boundaries when erroneous RC_BOC signals are detected. Setting this bit to one causes the ATMizer Architecture to ignore RC_BOC and assume that the Physical Layer does not lose synchronization

once the ACI Receiver is not in reset mode ($\overline{\text{RC\_RST}}$ deasserted).

CHOLD **Cell Holder** **[13:8]**

The Cell Holder Field contains the number (from 0 to 32) of receive cell holder buffers that still need to be processed by the APU.

RE **RC_BOC Error** **6**

The ACI Receiver sets this bit to one when it detects spurious RC_BOC (sourced by the Physical Layer). The APU must write to the System Control Register with Bit 6 set to one to clear the RC_BOC Error. To perform this write so that the other fields will not be affected, first read the System Control Register, OR this value with 0x00000040, and then store the result back to the System Control Register.

R **Regular Watchdog Timeout** **5**

When a slave does not reply with an acknowledgment after a certain time, the Host Bus Controller (external to the ATMizer Architecture) may take away the ATMizer Architecture's Bus Grant. At this time, the ATMizer Architecture's watchdog timer starts it's counter. If, after 64 clock cycles, the slave still does not reply with any kind of acknowledgment, then the Host/DMA Port asserts Interrupt0 and sets the Regular Watchdog Timeout Bit to one.

The APU must set the System Control Register Bit 5 to one to clear the Regular Watch Dog Timeout Interrupt. To perform this write so that the other fields will not be affected, first read the System Control Register, logically OR this value with 0x00000020, and then store this value back to the System Control Register.

Either the APU or the DMA can cause this timeout. When the APU causes a timeout, the Host/DMA Port asserts a bus error internal to the APU Core. This, in turn, creates an exception, and control branches to the exception vector. The APU sets a bit in the CP0 Cause Register (refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual*). When the APU causes a bus error, the Host/DMA Port does not set the Watchdog Timeout Bit in the System Control Register.

A **Atomic Watchdog Timeout** **4**

The Watchdog Timer sets the Atomic Watchdog Timeout Bit to one when it times out.

When the APU is performing an atomic transaction, the Host/DMA Port does not deassert HBS_RQ until the transaction is complete. If the last acknowledgment of the first transaction is received, and the APU does not start the second transaction within 64 clock cycles, then the Atomic Watchdog Timer times out and the Host/DMA Port asserts the Interrupt0 signal to the APU.

The APU must set the System Control Register Bit 4 to one to clear the Atomic Watch Dog Timeout Interrupt caused by an atomic transaction. To perform this write so that the other fields will not be affected, first read the System Control Register, logically OR this value with 0x00000010, and then store this value back to the System Control Register.

TAF **Transmit Address FIFO** **[2:0]**

This field indicates the status of the Assigned Cell address FIFO. It tells the number of valid cell pointers remaining to be transmitted in the FIFO. For example, TAF = 010_2 means that there are two cells remaining to be transmitted by the ACI transmitter.

TAF	*Number of Addresses Left*
000_2	0
001_2	1
010_2	2
011_2	3
100_2	4

14.2 APU Core Registers

The ATMizer Architecture APU core is based upon LSI Logic's CW33300. This section includes the LR33300 Family Control Registers described in the *CW33300 Enhanced Self-Embedding Processor Core User's Manual*.

The following registers are used in exception handling and cache control:

- BIU/Cache Configuration Register
- Status Register

- Cause Register
- Bad Address Register
- Target Address Register
- Exception Program Counter Register
- Processor Revision Identifier Register
- Debug and Cache Invalidate Control Register

The following registers are used in program debugging:

- Breakpoint Program Counter Register
- Breakpoint Program Counter Mask Register
- Breakpoint Data Address Register
- Breakpoint Data Address Mask Register

BIU/Cache Configuration Register

The APU BIU/Cache Configuration (BCC) Register allows software to configure both the APU Bus Interface Unit and the APU Cache Controller. This read/write register is 32 bits wide and has the following field definitions:

Location: Memory
Address: 0xFFFE0130 Reset Initial Value: 0x0

31 18	17	16	15	14	13	12	11	10	9 8	7	6	5 4	3	2	1	0
R	NOSTR	LDSCH	BGNT	NOPAD	RDPRI	INTP	IS1	IS0	IBLKSZ	DS	0	DBLKSZ	RAM	TAG	INV	LOCK

R **Reserved** **[31:18]**
These bits are reserved. Software should initialize these bits to zero to ensure compatibility with future versions of hardware.

NOSTR **No Streaming** **17**
NOT USED. This bit should always be cleared to zero.

LDSCH **Enable Load Scheduling** **16**
Setting this bit to one enables load scheduling. When load scheduling is enabled, the BIU determines whether a load is implemented immediately or, if the load data is not required yet, is delayed. Clearing this bit to zero disables load scheduling.

BGNT **Enable Bus Grant** **15**
NOT USED. This bit should always be inactive.

NOPAD **No Wait State** **14**
Clearing this bit to zero causes the APU to add a wait state at the end of every APU transaction. Setting this bit to one causes the APU to not add the wait state. Refer to Section 7.10, "Back-to-Back Operations," in the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for some functional waveforms, which show back-to-back operations with NOPAD enabled and disabled.

RDPRI **Enable Read Priority** **13**
Setting this bit to one makes APU load operations have priority over APU store operations.

INTP **Interrupt Polarity** **12**
NOT USED. This bit should always be inactive.

IS1 **Enable I-Cache Set 1** **11**
Setting this bit to one enables the IRAM.

IS0 **Enable I-Cache Set 0** **10**
This bit is cleared to zero.

IBLKSZ **I-Cache Refill Size** **[9:8]**
NOT USED. These bits should always be inactive.

DS **Enable D-Cache** **7**
Setting this bit to one enables the D-Cache.

DBLKSZ **D-Cache Refill Size** **[5:4]**
The DBLKSZ Field specifies the block size for data cache fill transactions as follows:

DBLKSZ		***Block Size (Words)***
0	0	2
0	1	4
1	0	Not used
1	1	Not used

Note that the data cache in the ATMizer Architecture is only four words deep, so the block size should be either two or four.

The BIU always attempts to fill the cache with a block fetch (internal signal BFREQ asserted). Devices that do not support block transactions deassert internal signal $\overline{\text{BFTCH}}$ HIGH, and the APU fetches only the missed word.

RAM **Scratchpad RAM** **3**
Setting this bit and the DS Bit to one causes the D-Cache to be used as a Scratchpad RAM. In Scratchpad RAM mode, the Cache Controller ignores the valid bits. The D-Cache is not filled when a load misses. Store operations that hit the D-Cache do not write to external memory.

TAG **Tag Test Mode** **2**
To enable tag test mode, both this bit and the Isolate Cache (IsC) Bit in the Status Register must be set to one. Refer to Section 5.4, "Cache Maintenance and Testing," in the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

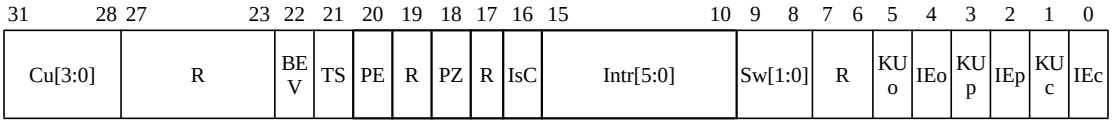
INV **Invalidate Mode** **1**
To enable invalidate mode, both this bit and the Isolate Cache (IsC) Bit in the Status Register must be set to one. Refer to Section 5.4, "Cache Maintenance and Testing," in the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

LOCK **Lock Mode** **0**
To enable lock mode, both this bit and the Isolate Cache (IsC) Bit in the Status Register must be set to one. Refer to Section 5.4, "Cache Maintenance and Testing," in the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for more information.

Status Register

The APU Status Register contains all major status bits for exception conditions. All bits in the Status Register, with the exception of the TS (TLB Shutdown) Bit, are readable and writable; the TS Bit is read-only. The format of the 32-bit Status Register is shown below. Additional details on the function of each Status Register Bit are provided in the paragraphs that follow.

Location: CP0 Cold Reset Initial Value: 0x00400000
Address: 12 Warm Reset Initial Value: 0x00400000



Cu[3:0] **Coprocessor Usability [3:0]** **[31:28]**

Software sets Cu[3:0] to one to indicate that the associated coprocessor is usable. Bit 31 corresponds to Coprocessor 3 and Bit 28 corresponds to Coprocessor 0. Because the APU does not support external coprocessors, Cu[3:1] should be cleared to zero, unless you intend to use the BCzF or BCzT instructions to test the internal CPC[3:0] signals. When Cu[3:1] are zero, a coprocessor instruction causes a Coprocessor Unusable Exception (CpU). Note that the system control coprocessor (CP0) is always considered usable when the APU is operating in kernel mode, regardless of the setting of the Cu0 Bit.

R **Reserved** **[27: 23], 19, 17, [7:6]**

These bits are reserved and read as zero. The APU ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of hardware.

BEV **Bootstrap Exception Vectors** **22**

This bit controls the location of general exception vectors during bootstrap (immediately following reset). When this bit is cleared to zero, the normal exception vectors are used; when the bit is set to one, bootstrap vector locations are used.

BEV Cleared to Zero – The debug exception vector is located at 0x80000040, and the general exception vector is located at 0x80000080.

BEV Set to One – The debug exception vector is relocated to an address of 0xBFC00140, and the general exception vector is relocated to 0xBFC00180. This alternate set of vectors can be used when diagnostic tests cause exceptions to occur prior to verification of proper operation of the cache and main memory system. The APU sets this bit to one upon deassertion of $\overline{\text{RESET}}$. (Refer to Section 8.3, “Exception Description Details,” in the *CW33300 Enhanced Self-Embedding Processor Core User’s Manual* for a description of the exception vectors.)

TS **TLB Shutdown** **21**
This bit is cleared to zero. This bit is read-only.

PE **Parity Error** **20**
The APU sets this bit to one if it detects an external memory parity error. Software may use PE to log external memory parity errors. The parity error condition is reset by writing a one into PE; writing a zero into this bit does not affect its value.

PZ **Parity Zero** **18**
Setting this bit to one forces the APU to generate zeros on internal signals DP[3:0] during store transactions.

IsC **Isolate Cache** **16**
When this bit is set to one, store operations do not propagate through the APU to the external memory system. This bit must be set to one for cache testing. Refer to Section 5.4, “Cache Maintenance and Testing,” in the *CW33300 Enhanced Self-Embedding Processor Core User’s Manual* for more information.

Intr[5:0] **Hardware Interrupt Mask [5:0]** **[15:10]**
Software sets these six bits to one to enable the corresponding hardware interrupts. Bit 15 corresponds to internal signal INT5, and Bit 10 corresponds to internal signal INT0. All interrupts can be disabled by clearing the Interrupt Enable Bits (IEo/IEp/IEc) described below.

Sw[1:0] **Software Interrupt Mask [1:0]** **[9:8]**
Software sets these two bits to one to enable the corresponding software interrupts. Bit 9 corresponds to Sw1, and Bit 8 corresponds to Sw0. All interrupts can be disabled by clearing the Interrupt Enable Bits (IEo/IEp/IEc) described below.

KUo, p, c **Kernel-User Mode, Old/Previous/Current** **5, 3, 1**
The KUo, KUp, and KUc bits comprise a three-level stack showing the old/previous/current mode (0 means kernel; 1 means user). The occurrence of an exception automatically puts the system in kernel mode. Manipulation and use of these bits during exception processing is described in the following section.

IEo, p, c **Interrupt Enable, Old/Previous/Current** **4, 2, 0**
The IEo, IEp, and IEc bits comprise a three-level stack showing the old/ previous/current interrupt enable settings (0 means disabled; 1 means enabled). Manipulation and use of these bits during exception processing is described in the following section.

Status Register Mode Bits and Exception Processing

When the APU responds to an exception, it saves the *current* kernel/user mode (KUc) and *current* interrupt enable mode (IEc) bits of the Status Register into the *previous* mode bits (KUp and IEp). The *previous* mode bits (KUp and IEp) are saved into the old mode bits (KUo and IEo). The current mode bits (KUc and IEc) are cleared to cause the processor to enter the kernel operating mode and to disable all interrupts.

This three-level set of mode bits lets the APU respond to two levels of exceptions before software must save the contents of the Status Register. Figure 14.2 shows how the APU manipulates the Status Register during exception recognition.

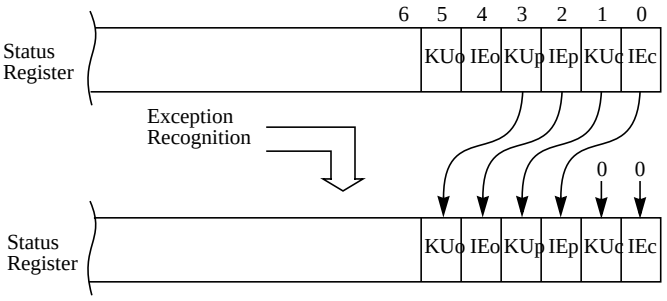
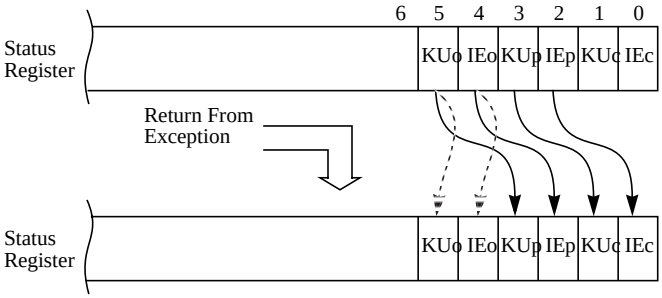


Figure 14.2 The Status Register and Exception Recognition

After an exception handler has completed execution, the APU must return to the system context that existed prior to the exception (if possible). The Restore From Exception (RFE) instruction provides the mechanism for this return.

The RFE instruction restores control to a process that was preempted by an exception. When the RFE instruction is executed, it restores the *previous* Interrupt Mask (IEp) Bit and Kernel/user Mode (KUp) Bit in the Status Register into the corresponding *current* status bits (IEc and KUc). It also restores the *old* status bits (IEo and KUo) into the corresponding previous status bits (IEp and KUp). The old status bits (IEo and KUo) remain unchanged. The actions of the RFE instruction are illustrated in Figure 14.3.

Figure 14.3 Restoring from Exceptions

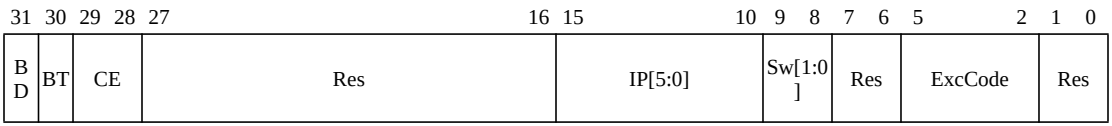


Cause Register

The contents of the Cause Register describe the last occurring exception. A four-bit exception code field (ExcCode) indicates the cause of the exception. The remaining fields contain detailed information specific to certain exceptions.

With the exception of the Sw Bits, all bits in the Cause Register are read-only. Writes to the Sw Bits set or reset software interrupts. The format of the 32-bit Cause Register is shown below.

Location: CP0 Cold Reset Initial Value: 0x00000000
Address: 13 Warm Reset Initial Value: 0x00000000



BD **Branch Delay** **31**
The APU sets this bit to one to indicate that the last exception was taken while executing in a branch delay slot.

BT **Branch Taken** **30**
When the BD Bit is set, the BT Bit determines whether or not the branch is taken. A value of one in BT indicates that the branch is taken. The Target Address Register holds the return address.

CE **Coprocessor Error** **[29:28]**
When taking a Coprocessor Unusable exception, the APU writes the references coprocessor number in this field. This field is otherwise undefined.

Res **Reserved** **[27:16], [7:6], [1:0]**
These bits are reserved and read as zero. The APU ignores attempts to set these bits; however, software should write these

bits as zero to ensure compatibility with future versions of hardware.

IP[5:0] **Interrupt Pending [5:0]** **[15:10]**
The APU sets these bits to indicate that an external interrupt is pending on INT[5:0]. Bit 15 corresponds to INT5 and Bit 10 corresponds to INT0.

Sw[1:0] **Software Interrupts [1:0]** **[9:8]**
By setting either of these bits to one, software causes the APU to transfer control to the general exception routine. Bit 9 corresponds to Sw1. The exception routine can tell which software interrupt bit is set by reading this field. The exception routine must reset the Sw Bits to zero before returning control to the interrupting software.

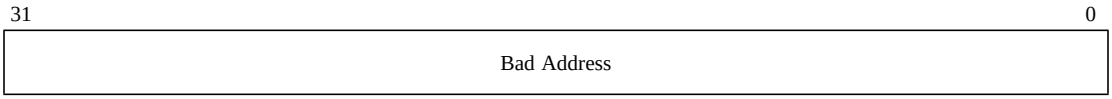
ExcCode **Exception Code** **[5:2]**
The APU sets this field to indicate the type of event that caused the last general exception. The four bits are encoded as described in the table below.

Value	*Mnemonic*	*Description*
0	Int	External Interrupt
1	—	Reserved
2	—	Reserved
3	—	Reserved
4	AdEL	Address Error Exception (load or instruction)
5	AdES	Address Error Exception (store)
6	IBE	Bus Error Exception (for an instruction fetch)
7	DBE	Bus Error Exception (for a data load or store)
8	Sys	SYSCALL Exception
9	Bp	Breakpoint Exception
A	RI	Reserved Instruction Exception
B	CpU	Coprocessor Unusable Exception
C	Ovf	Arithmetic Overflow Exception
D-F	—	Reserved

Bad Address Register

The Bad Address (BadA) Register is a read-only register that saves the address associated with an illegal access. This register saves only addressing errors (*AdEL* or *AdES*), not bus errors. The format of the 32-bit BadA Register is shown below.

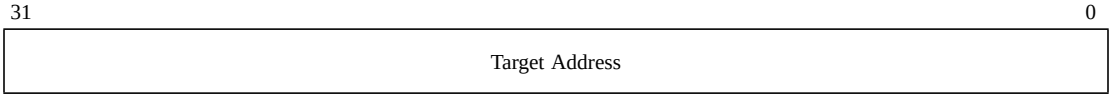
Location: CP0 Cold Reset Initial Value: Undefined
Address: 8 Warm Reset Initial Value: Unchanged



Target Address Register

The Target Address (TAR) Register is a read-only register that holds the return address for a branch. When the cause of an exception is in the branch delay slot (the APU sets the BD Bit in the Cause Register to one), execution resumes either at the target of the branch or at the Exception Program Counter [EPC] + 8. If the branch was taken, the APU sets the BT Bit in the Cause Register to one and loads the branch target address in the TAR Register. The exception handler needs only to load this address into a register and jump to that location. The format of the 32-bit TAR Register is shown below.

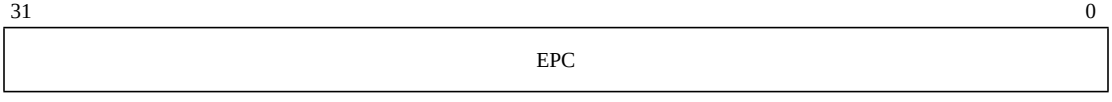
Location: CP0 Cold Reset Initial Value: Undefined
Address: 6 Warm Reset Initial Value: Unchanged



Exception Program Counter Register

The 32-bit Exception Program Counter (EPC) Register contains the address where processing resumes after an exception is serviced. In most cases, the EPC Register contains the address of the instruction that caused the exception. However, when the exception instruction resides in a branch delay slot, the Cause Register's BD Bit is set to one to indicate that the EPC Register contains the address of the immediately preceding branch or jump instruction. The format of the EPC Register is shown below.

Location: CP0 Cold Reset Initial Value: Undefined
Address: 14 Warm Reset Initial Value: Unchanged



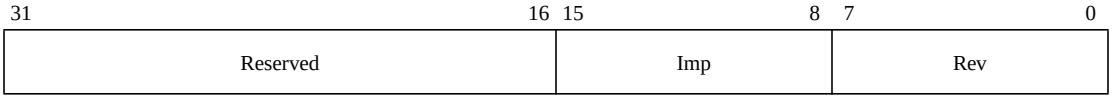
Processor Revision Identifier Register

The Processor Revision Identifier (PRId) Register contains information that identifies the implementation and revision level of the processor and system control coprocessor.

The revision number can distinguish some chip revisions. However, LSI Logic is free to change this field at any time and does not guarantee that changes to its chips necessarily change the revision number or that changes to the revision number necessarily reflect real chip changes. For this reason, software should not rely on the revision number to characterize the chip.

The format of this 32-bit, read-only register is shown below.

Location: CP0 APU Reset Initial Value: 0x00000AXX
Address: 15



Reserved **Reserved** **[31:16]**
These bits are reserved and read as zero. The APU ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of hardware.

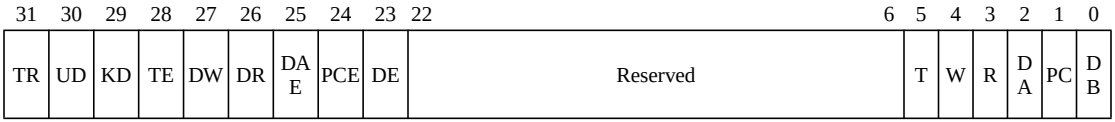
Imp **Implementation** **[15:8]**
This eight-bit field contains the APU's implementation number, 0x0A.

Rev **Revision** **[7:0]**
This eight-bit field contains the APU's revision number, 0x0A.

Debug and Cache Invalidate Control Register

The Debug and Cache Invalidate Control (DCIC) Register contains the enable and status bits for the APU's breakpoint mechanism and the control bits for the Cache Controller's invalidate mechanism. All bits in the DCIC Register are readable and writable. The format of the DCIC Register is shown below. Additional details on the function of each DCIC Register Bit are provided in the paragraphs that follow.

Location: CP0 Cold Reset Initial Value: 0x00000000
Address: 7 Warm Reset Initial Value: 0x00000000



Breakpoint Control Bits – There are three types of breakpoint exceptions: those caused by instruction fetches (break on program counter), those caused by data accesses (break on data address), and those caused by non-sequential instruction fetches (trace). To make the debug mechanism precise, it can be configured to detect breakpoints in user mode or kernel mode or both. It can also be configured to discriminate between read and write accesses. All breakpoint and trace conditions are posted as Breakpoint Exceptions in the Cause Register.

TR **Trap Enable** **31**
When software sets TR to one, the APU vectors to the Debug Exception address when it encounters a debug condition. If this bit is zero, the APU will not trap, but it will set the appropriate debug status bits for any debug condition that software has enabled.

UD **User Debug Enable** **30**
When software sets UD to one, the APU detects debug conditions when running in user mode.

KD **Kernel Debug Enable** **29**
When software sets KD to one, the APU detects debug conditions when running in kernel mode.

TE **Trace Enable** **28**
When software sets TE to one and the APU fetches any non-sequential instruction, the APU vectors to the Debug Exception address.

Three bits control the APU's Breakpoint Data Address feature. DAE enables or disables the breakpoint function. DW and DR control whether read and/or write accesses are trapped.

DW **Data Write Enable** **27**
When software sets DW and DAE to one and the APU writes to the address specified in the Breakpoint Data Address Register, the APU vectors to the Debug Exception Address.

DR	**Data Read Enable** When software sets DR and DAE to one and the APU reads from the address specified in the Breakpoint Data Address Register, the APU vectors to the Debug Exception Address.	**26**
DAE	**Data Address Breakpoint Enable** When software sets DAE to one and the APU accesses the address specified in the Breakpoint Data Address Register, the APU vectors to the Debug Exception Address.	**25**
PCE	**Program Counter Breakpoint Enable** When software sets PCE to one and the APU fetches an instruction from the address specified in the Breakpoint Program Counter Register, the APU vectors to the Debug Exception Address.	**24**
DE	**Debug Enable** Software sets DE to one to enable the APU's debug facility. If this bit is zero, the APU will not detect the breakpoint or trace conditions specified in the DCIC Register, Bits [31:24].	**23**
Reserved	**Reserved** These bits are reserved and read as zero. The APU ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of hardware.	**[22:6]**

Breakpoint Status Bits – The breakpoint mechanism posts the cause of a Debug Exception with these six status bits.

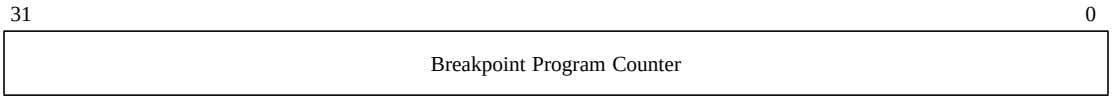
T	**Trace** The APU sets T to one when it detects a trace condition.	**5**
W	**Write Reference** The APU sets W to one when it detects a write reference to the address specified in the Breakpoint Address Register.	**4**
R	**Read Reference** The APU sets R to one when it detects a read reference to the address specified in the Breakpoint Data Address Register.	**3**
DA	**Data Address** The APU sets DA to one when it detects a data address debug condition.	**2**

PC	**Program Counter** The APU sets PC to one when it detects a program counter debug condition.	**1**
DB	**Debug** The APU sets DB to one when it detects any debug condition.	**0**

Breakpoint Program Counter Register

The Breakpoint Program Counter (BPC) Register is a read/write register that software uses to specify a program counter breakpoint. The format of the 32-bit BPC Register is shown below.

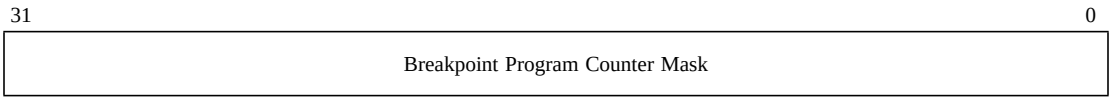
Location: CP0 — Cold Reset Initial Value: Undefined
Address: 3 — Warm Reset Initial Value: Unchanged



Breakpoint Program Counter Mask Register

The Breakpoint Program Counter Mask (BPCM) Register is a read/write register that masks bits in the BPC Register. A one in any bit in the BPCM Register indicates that the APU compares the corresponding bit in the BPC Register for program counter exceptions. Values of zero in the mask indicate that the APU does not check the corresponding bits in the BPC Register for exceptions. The format of the 32-bit BPCM Register is shown below.

Location: CP0 — Cold Reset Initial Value: 0xFFFFFFFF
Address: 11 — Warm Reset Initial Value: 0xFFFFFFFF

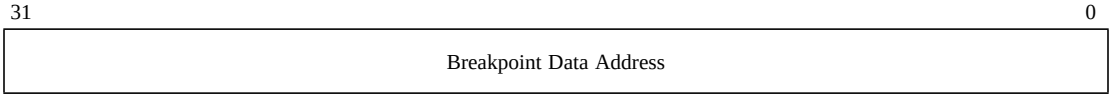


For example, if the BPCM Register is set to 0xFFFF0000, the APU compares Bits [31:16] of the Program Counter with the corresponding bits in the BPC Register for program counter exceptions.

Breakpoint Data Address Register

The Breakpoint Data Address (BDA) Register is a read/write register that software uses to specify a data address breakpoint. The format of the 32-bit BDA Register is shown below.

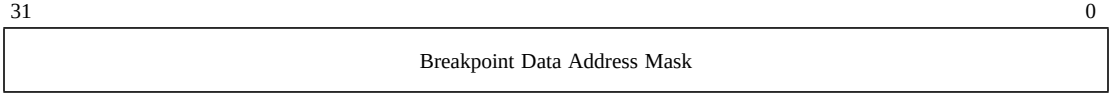
Location: CP0
Address: 5
Cold Reset Initial Value: Undefined
Warm Reset Initial Value: Unchanged



Breakpoint Data Address Mask Register

The Breakpoint Data Address Mask (BDAM) Register is a read/write register that masks bits in the BDA Register. A one in any bit in the BDAM Register indicates that the APU compares the corresponding bit in the BDA Register for debug exceptions. Values of zero in the mask indicate that the APU does not check the corresponding bits in the BDA Register for exceptions. The format of the 32-bit BDAM Register is shown below.

Location: CP0
Address: 9
Cold Reset Initial Value: 0xFFFFFFFF
Warm Reset Initial Value: 0xFFFFFFFF



For example, if the BDAM Register is set to 0xFFFF0000, the APU compares Bits [31:16] of the BDA Register for debug exceptions.

14.3 Other Registers Summary

This section contains register summaries (and a cross reference) of all the other registers contained in the ATMizer Architecture.

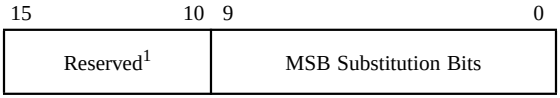
Host Interrupt Register

The Host Interrupt Register is not really a register. It is an address decode circuit with an Effective Address of 0xFFF04B00. (See Section 4.7, “ATMizer Architecture-to-Host Messaging,” on page 4-5.)

MSB Substitution Register

Figure 14.4 shows the MSB Substitution Register format (also shown on page 4-7).

Figure 14.4 MSB Substitution Register



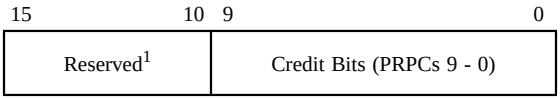
1. All reserved bits must be 0.

Effective Address = 0xFFF04D00

PRU Channel Group Credit Register

Figure 14.5 shows the PRU Channel Group Credit Register (CGCR) format (also shown on page 7-3).

Figure 14.5 Channel Group Credit Register



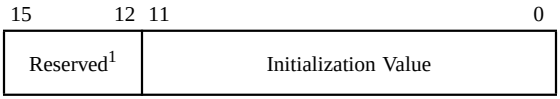
1. All reserved bits must be 0.

Effective Address = 0xFFF040X0

PRU 12-Bit Count Initialization Registers

Figure 14.6 shows the PRU 12-bit Count Initialization Register (CIR) format used for PRPCs 0 - 7 (also shown on page 7-3).

Figure 14.6 12-Bit Count Initialization Register



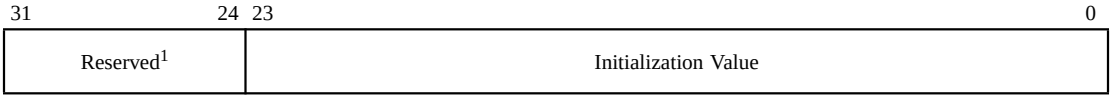
1. All reserved bits must be zero.

Effective Addresses = 0xFFF043XX (see Table 11.8)

PRU 24-Bit Count Initialization Registers

Figure 14.7 shows the PRU 24-bit Count Initialization Register format used for PRPCs 8 - 9 (also shown on page 7-4).

Figure 14.7
24-Bit Count
Initialization Register



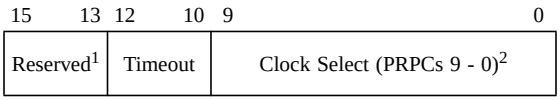
1. All reserved bits must be 0.

Effective Addresses = 0xFFF043XX (see Table 11.8)

PRU Configuration Register

Figure 14.8 shows the PRU Configuration Register (CR) format (also shown on page 7-5).

Figure 14.8
Configuration Register



1. All reserved bits must be 0.
2. CLK, if bit cleared to zero. PRU_CLK, if bit set to one.

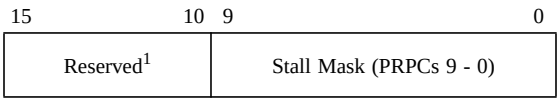
Effective Address = 0xFFF04100

Note Firmware must *not* write to the Configuration Register using a Store Half-word instruction.

PRU Stall Register

Figure 14.9 shows the PRU Stall Register (SR) format (also shown on page 7-6).

Figure 14.9
Stall Register



1. All reserved bits must be 0.

Effective Address = 0xFFF04200

DMAC Control Register

Figure 14.10 shows the format of the DMAC Control Register (also shown on page 8-4).

Figure 14.10
DMAC Control Register

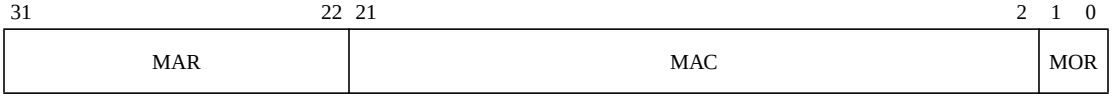
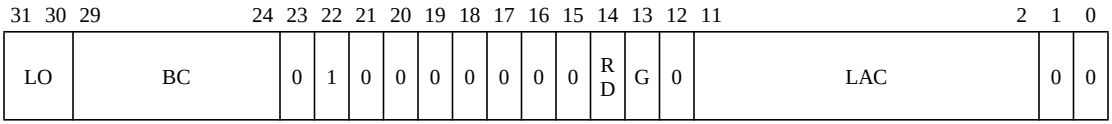


Figure 14.11 shows the format of the Effective Address of the DMAC Control Register (also shown on page 8-2). For more information see also Section 11.1, “Memory Maps.”

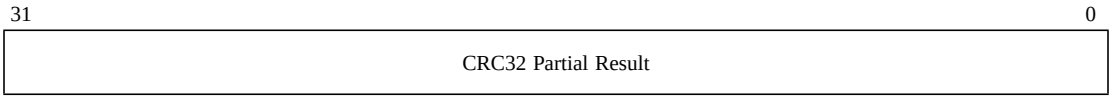
Figure 14.11
DMAC Control Register’s Effective Address



CRC32 Register

Figure 14.12 shows the format of the CRC32 Register (also shown on page 8-5).

Figure 14.12
CRC32 Register

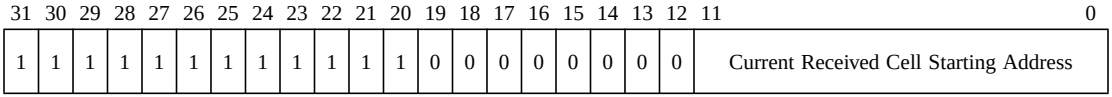


Effective Address = 0xFFF04CX0

ACI Current Received Cell Address Register

Figure 14.13 shows the ACI Current Received Cell Address Register (also shown on page 9-10).

Figure 14.13
Current Received Cell Address Register



Effective Address = 0xFFF04400

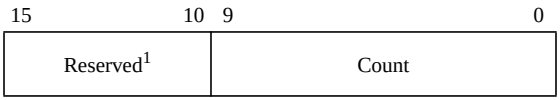
Received Cell Indicator Register

The Received Cell Indicator Register is not really a register. It is an address decode circuit with an Effective Address of 0xFFF0460C. (See subsection entitled “Received Cell Indication”on page 9-9.)

ACI Global Pacing Rate Register

Figure 14.14 shows the ACI Global Pacing Rate Register (also shown on page 9-13). The Store Halfword instruction should be used to write to this register.

Figure 14.14
Global Pacing Rate Register



1. All reserved bits must be 0.

Effective Address = 0xFFF047X0

Chapter 15
Specifications

This chapter describes the electrical specifications for the LSI Logic L64360 chip, based upon the ATMizer Architecture.

This chapter has four sections:

- Section 15.1, “AC Timing”
- Section 15.2, “Electrical Requirements”
- Section 15.3, “Pin Summary”
- Section 15.4, “Pinout, Pin List, and Package Information”

15.1 AC Timing

This section specifies the AC timing characteristics of the L64360. The relationship between various signals is depicted in Figures 15.3 through 15.9. The figures depict:

- Secondary Port Timing
- Host/DMA Port Timing 1
- Host/DMA Port Timing 2
- Transmitting Cell Timing
- ACI Transmitter TX_IDLE Timing
- Received Cell Timing
- ACI Receiver RC_FULL Timing

Table 15.1 shows AC timing values specified for 70 pF loading. The numbers in Figures 15.3 through 15.9 refer to the AC timing parameters listed in Table 15.1.

The L64360 was designed using LSI Logic’s LEA300K process.

During AC testing, HIGH inputs are driven at V_{DD}, and LOW inputs are driven at 0 V. For transitions between HIGH, LOW, and invalid states, timing measurements are made at 1.5 V, as shown in Figure 15.1.

For 3-state outputs, timing measurements are made from the point at which the output turns ON or OFF. An output is ON when its voltage is greater than 3.5 V or less than 1.5 V. An output is OFF when its voltage is less than V_{DD} − 1.5 V or greater than 1.5 V, as shown in Figure 15.2.

Figure 15.1
AC Test Load and Waveform for Standard Outputs

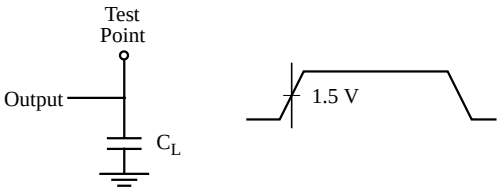


Figure 15.2
AC Test Load and Waveform for 3-State Outputs

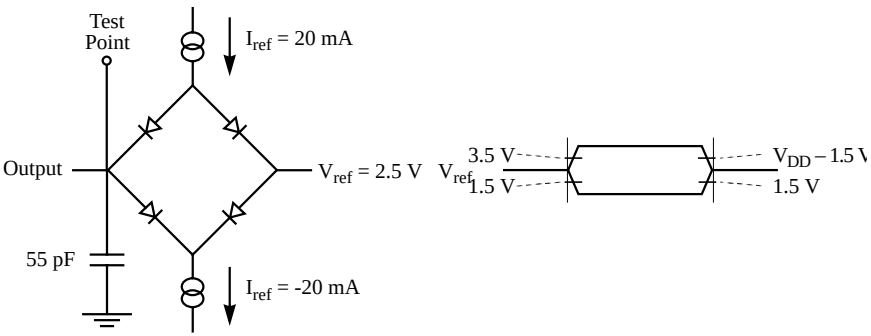


Table 15.1
AC Timing Values for
70-pF Loading (in ns)

		50 MHz		40 MHz		33 MHz		25 MHz	
	Description	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
1	CLK High/Low Time	9.6		12		14		17	
2	Output Delay from Rising CLK to SP_RQ High or Low		14		16		17		17
3	Output Delay from Rising CLK to SP_WR Valid		13		15		16		16
4	Output Delay from SP_GNT High to SP_AD[31:0] Address Valid		11		13		14		14
5	Output Delay from SP_ASEL Low to SP_AD[31:0] Data Valid for Write		11		13		14		14
6	Output Delay from SP_ASEL Low to SP_AD[31:0] 3-State for Read		11		13		14		14
7	Input Setup from $\overline{\text{SP\_ACK}}$ Low to Rising CLK	3		5		5		6	
8	Input Setup from $\overline{\text{SP\_BWIDE}}$ Low to Rising CLK	3		5		5		6	
9	Input Setup from SP_AD[31:0] Data Valid to Rising CLK for Read	6		8		8		9	
10	Input Hold from Rising CLK to SP_AD[31:0] Data Invalid for Read	2		3		3		4	
11	Output Delay from Rising CLK to HBS_RQ High		11		13		14		14
12	Output Delay from Rising CLK to HBS_WR Valid		15		17		18		18
13	Output Delay from Rising CLK to $\overline{\text{HBS\_AS}}$ Low		15		17		18		18
14	Output Delay from Rising CLK to $\overline{\text{HBS\_END}}$ Low		15		17		18		18
15	Output Delay from Rising CLK to HBS_A[31:2] Valid		15		17		18		18
16	Output Delay from Rising CLK to HBS_D[31:0] Valid for Write		15		17		18		18
17	Output Delay from Rising CLK to $\overline{\text{HBS\_BE}}$[3:0] Valid		15		17		18		18
18	Output Delay from Rising CLK to HBS_S[2:0] Valid		15		17		18		18
19	Output Delay from Rising CLK to HBS_WR 3-State		13		15		16		16
20	Output Delay from Rising CLK to $\overline{\text{HBS\_AS}}$ 3-State		13		15		16		16
21	Output Delay from Rising CLK to $\overline{\text{HBS\_END}}$ 3-State		13		15		16		16
22	Output Delay from Rising CLK to HBS_A[31:2] 3-State		14		16		17		17
23	Output Delay from Rising CLK to HBS_D[31:0] 3-State for Write		14		16		17		17

(Sheet 1 of 3)

Table 15.1 (Cont.)
AC Timing Values for
70-pF Loading (in ns)

		50 MHz		40 MHz		33 MHz		25 MHz	
	Description	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
24	Output Delay from Rising CLK to $\overline{\text{HBS\_BE}}$[3:0] 3-State		14		16		17		17
25	Output Delay from Rising CLK to HBS_S[2:0] 3-State		14		16		17		17
26	Input Setup from HBS_GNT High to Rising CLK	3		5		5		6	
27	Input Setup from $\overline{\text{HBS\_ACK}}$ Low to Rising CLK	7		9		9		10	
28	Input Setup from HBS_D[31:0] Valid to Rising CLK	4		6		6		7	
29	Input Hold from Rising CLK to HBS_GNT Low	2		3		3		4	
30	Input Hold from Rising CLK to $\overline{\text{HBS\_ACK}}$ Low	2		3		3		4	
31	Input Hold from Rising CLK to HBS_D[31:0] Invalid for Read	2		3		3		4	
32	Output Delay from HBS_AOE High to HBS_A[31:2] Valid		14		16		17		17
33	Output Delay from HBS_AOE Low to HBS_A[31:2] 3-State		14		16		17		17
34	Output Delay from HBS_DOE High to HBS_D[31:0] Valid for Write		14		16		17		17
35	Output Delay from HBS_DOE Low to HBS_D[31:0] 3-State for Write		14		16		17		17
36	Output Delay from Rising CLK to HBS_INT High		12		14		15		15
37	TX_CLK Cycle Time	40		50		60		80	
38	Output Delay from Rising TX_CLK to $\overline{\text{TX\_RST}}$ High		12		14		15		15
39	Output Delay from Rising TX_CLK to $\overline{\text{TX\_DRDY}}$ Low or High		9		11		12		12
40	Output Delay from Rising TX_CLK to TX_BOC High		13		15		16		16
41	Output Delay from Rising TX_CLK to TX_D[7:0] Valid		15		17		18		18
42	Input Setup from TX_ACK Valid to Rising TX_CLK	7		9		9		9	
43	Input Hold from Rising TX_CLK to TX_ACK Valid	1		2		2		3	
44	Input Setup from $\overline{\text{TX\_FULL}}$ Low to Rising TX_CLK	7		9		9		9	
45	Input Hold from Rising TX_CLK to $\overline{\text{TX\_FULL}}$ High	1		2		2		3	
46	Output Delay from Rising TX_CLK to TX_IDLE Valid		9		11		12		12
47	RC_CLK Cycle Time	40		50		60		80	
48	Output Delay from Rising RC_CLK to $\overline{\text{RC\_RST}}$ High		10		12		13		13

(Sheet 2 of 3)

Table 15.1 (Cont.)
AC Timing Values for
70-pF Loading (in ns)

		50 MHz		40 MHz		33 MHz		25 MHz	
	Description	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**	**Min**	**Max**
49	Input Setup from RC_BOC High to Rising RC_CLK	7		9		9		10	
50	Input Setup from RC_ACK High or Low to Rising RC_CLK	7		9		9		10	
51	Input Setup from RC_D[7:0] Valid to Rising RC_CLK	7		9		9		10	
52	Input Hold from Rising RC_CLK to RC_D[7:0] Invalid	1		2		2		3	
53	Input Hold from Rising RC_CLK to RC_ACK Low or High	1		2		2		3	
54	Input Hold from Rising RC_CLK to RC_BOC Low	1		2		2		3	
55	Output Delay from Rising RC_CLK to HEC_ERR High		10		12		13		13
56	Output Delay from Rising RC_CLK to RC_FULL High or Low		10		12		13		13

(Sheet 3 of 3)

Figure 15.3
Secondary Port Timing

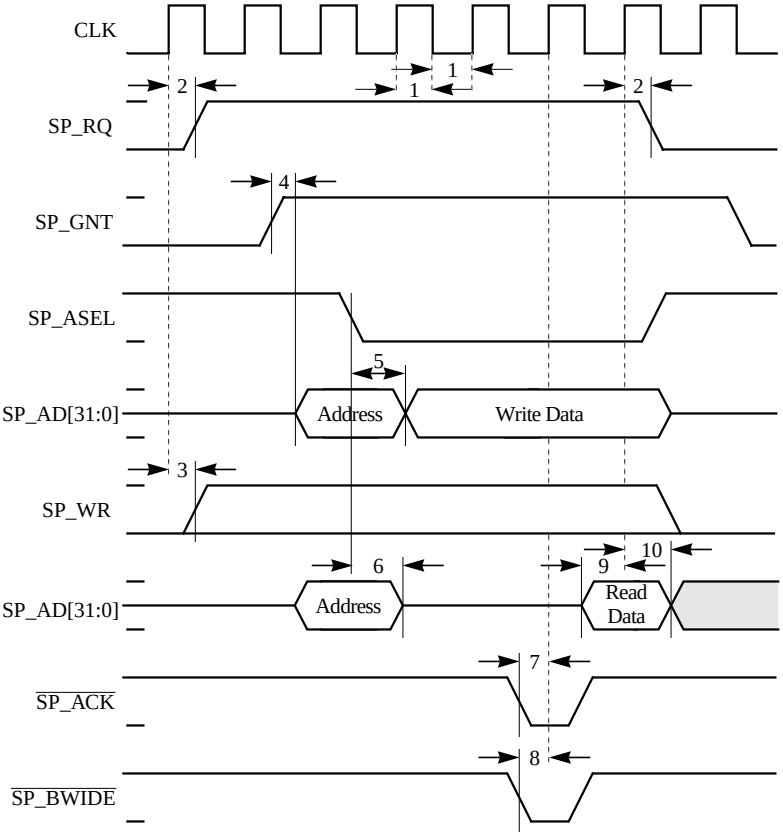


Figure 15.4
Host/DMA Port Timing 1

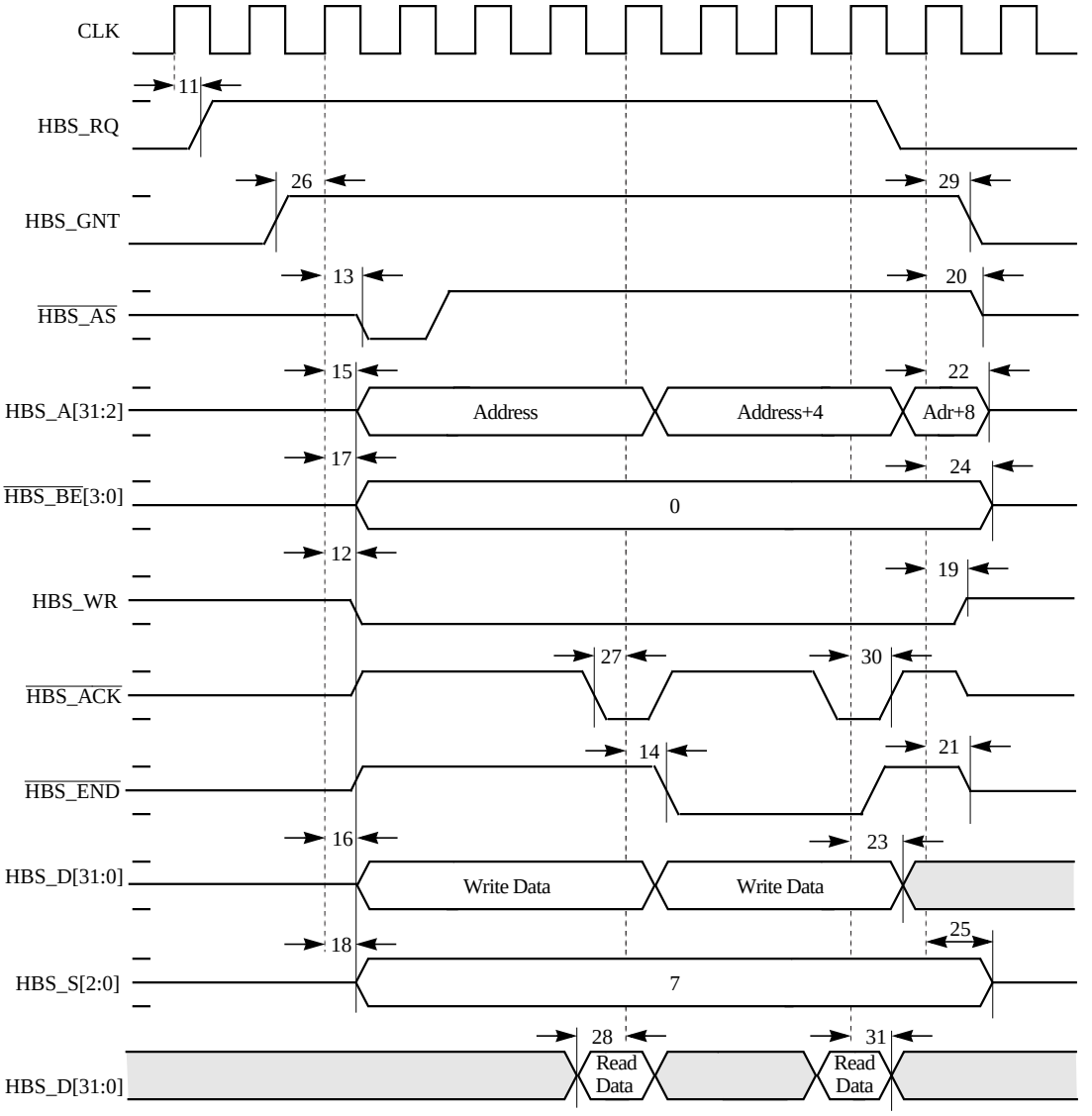


Figure 15.5
Host/DMA Port Timing 2

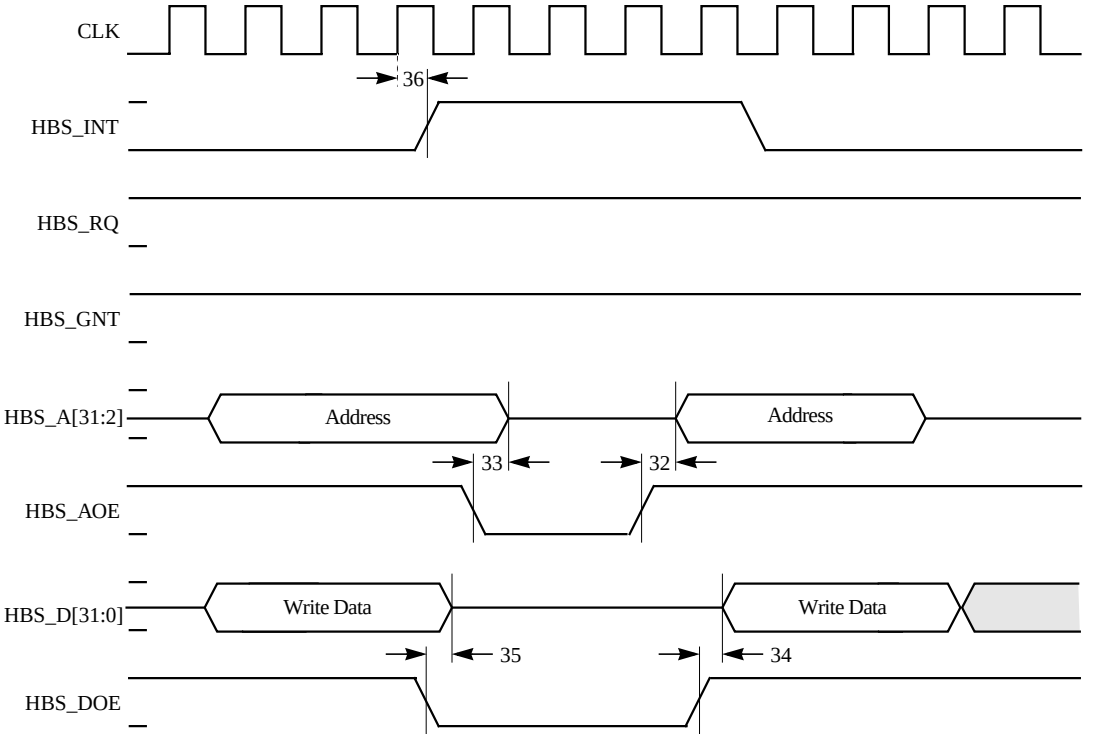


Figure 15.6
Transmitting Cell Timing

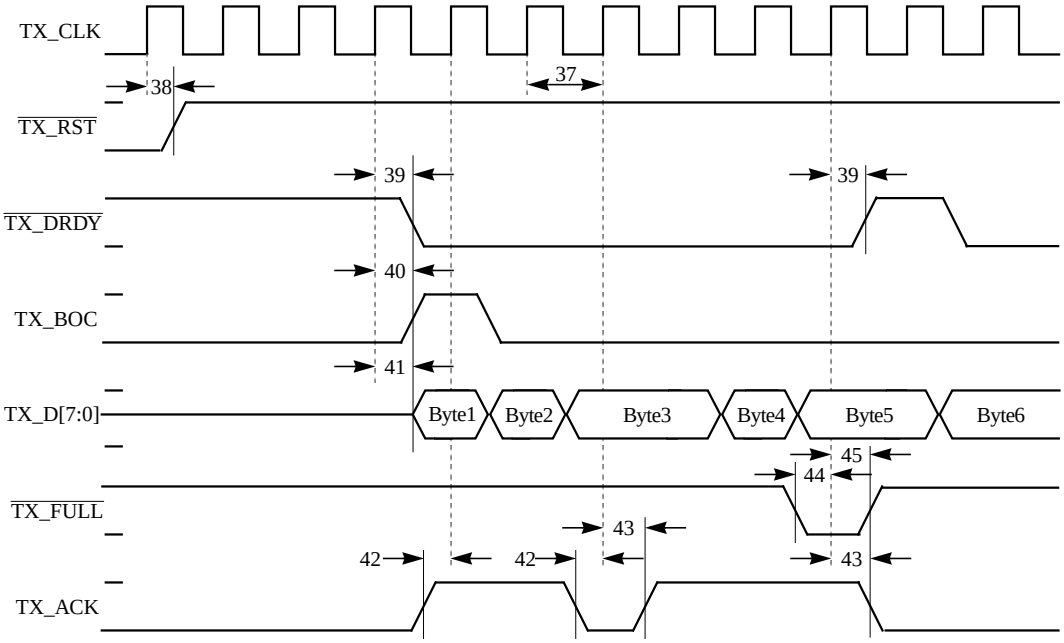


Figure 15.7
ACI Transmitter
TX_IDLE Timing

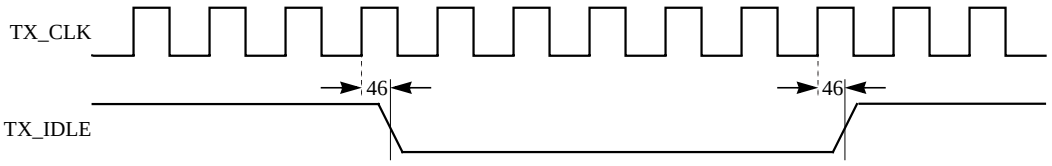


Figure 15.8
Received Cell Timing

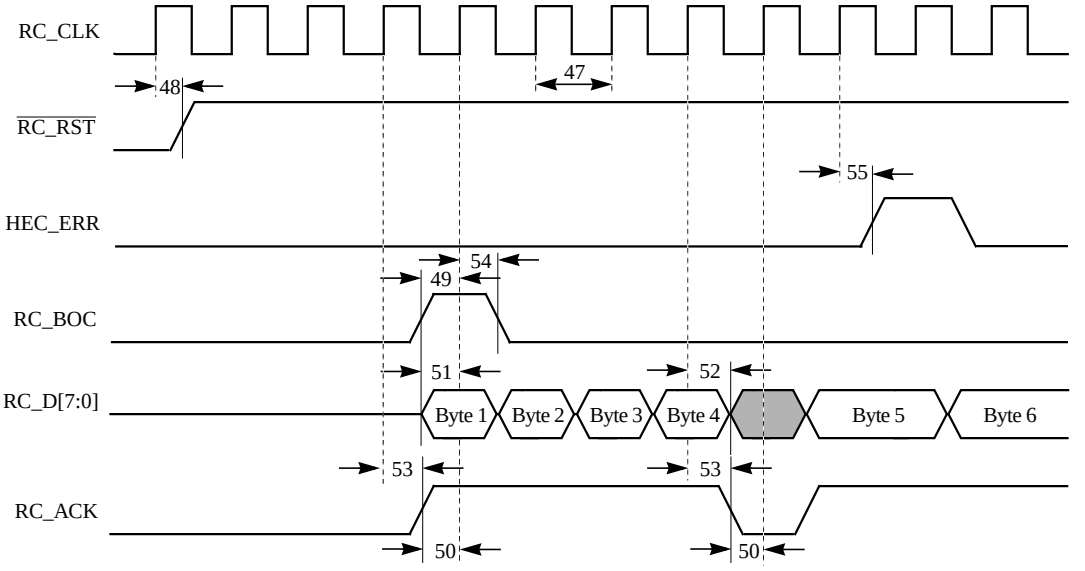
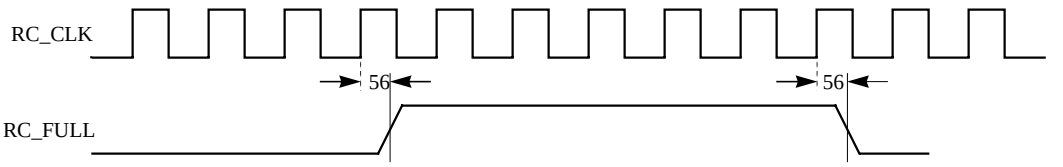


Figure 15.9
ACI Receiver RC_FULL Timing



15.2 Electrical Requirements

This section specifies the electrical requirements for the L64360. Five tables list electrical data in the following categories:

- Absolute Maximum Ratings
- Recommended Operating Conditions
- Worst Case Thermal Operating Conditions
- Capacitance
- DC Characteristics

Table 15.2 Absolute Maximum Ratings

Symbol	Parameter	Limits[1]	Unit
V_{DD}	DC Supply	-0.3 to +7.0	V
V_{IN}	Input Voltage	-0.3 to V_{DD} + 0.3	V
I_{IN}	DC Input Current	10	mA
T_{STG}	Storage Temperature Range, metal	0 to +125	°C

1. Referenced to V_{SS}.

Table 15.3 Recommended Operating Conditions

Symbol	Parameter	Limits	Unit
V_{DD}	DC Supply	+ 4.75 to +5.25	V
T_A	Ambient Temperature (50 MHz)	0 to 50	°C
T_A	Ambient Temperature (40 MHz and Below)	0 to 70	°C

Table 15.4 Worst Case Thermal Operating Conditions

Operating Speed	Case Temperature	Unit
50 MHz	0 to 85	°C
40 MHz and Below	0 to 100	°C

Table 15.5 Capacitance

Symbol	Parameter[1]	Min	Typ	Max	Unit
C_{IN}	Input Capacitance			5	pF
C_{OUT}	Output Capacitance			10	pF
C_{IO}	I/O Bus Capacitance			15	pF

1. Measurement conditions are V_{IN} = 5.0 V, T_A = 25° C, and clock frequency = 1 MHz.

Table 15.6 DC Characteristics

Symbol	Parameter	Condition[1]	Min	Typ	Max	Units
V_{IL}	Voltage Input Low		–	–	0.8	V
V_{IH}	Voltage Input High		2.0	–	–	V
V_{OH}	Voltage Output High	I_{OH} = -4.0 mA	2.4	4.5	–	V
		I_{OH} = -8.0 mA	2.4	4.5	–	V
V_{OL}	Voltage Output Low	I_{OL} = 4.0 mA	–	0.2	0.4	V
		I_{OL} = 8.0 mA	–	0.2	0.4	V
I_{IL}	Current Input Leakage	V_{DD} = Max, V_{IN} = V_{DD} or V_{SS}	-10	±1	10	µA

Table 15.6 (Cont.)
DC Characteristics

Symbol	*Parameter*	*Condition*[1]	*Min*	*Typ*	*Max*	*Units*
I_{OZ}	Current 3-State Output Leakage	V_{DD} = Max, V_{OUT} = V_{SS} or V_{DD}	-10	±1	10	µA
I_{IPU}	Current Input Pull-up	V_{IN} = V_{SS}	-35	-115	-350	µA
I_{OZU}	Current 3-State Output w/Pull-up	V_{IN} = V_{SS}	-2	–	-175	µA
I_{DD}	Quiescent Supply Current	V_{IN} = V_{DD} or V_{SS}	–	–	100	µA
I_{CC}	Dynamic Supply Current	V_{DD} = Max, 25 MHz	–	–	480	mA
		V_{DD} = Max, 33 MHz	–	–	550	mA
		V_{DD} = Max, 40 MHz	–	–	650	mA
		V_{DD} = Max, 50 MHz	–	–	750	mA

1. Specified at V_{DD} = 5 ± 5% at ambient temperature over the ranges specified in Table 15.3.

15.3 Pin Summary

Table 15.7 summarizes the L64360 pins. The table provides the drive capacity of the outputs and the signal types for both output and input pins.

Table 15.7 L64360 Pin Description Summary

Mnemonic	*Description*	*Type*	*Drive (mA)*	*Active*
CLK	System Clock	Input	4	–
GPINT_AUTO	General Purpose APU Interrupt	Input	4	High
GPINT_TST	L64360 Interrupt	Input	4	High
HBS_A[31:2]	Host/DMA Port Address Bus	Output	4	–
$\overline{\text{HBS\_ACK}}$	Host/DMA Port Data Acknowledgment	Input	4	Low
HBS_AOE	Host/DMA Port Address Output Enable	Input	4	High
$\overline{\text{HBS\_AS}}$	Host/DMA Port Address Strobe	Output	4	Low
$\overline{\text{HBS\_BE}}$[3:0]	Host/DMA Port Byte Enables	Output	4	Low
HBS_BOOT	Host/DMA Port Boot Select	Input	4	High
HBS_D[31:0]	Host/DMA Port Data Bus	Bidirectional	4	–
HBS_DOE	Host/DMA Port Output Enable	Input	4	High
$\overline{\text{HBS\_END}}$	Host/DMA Port Operation Ending	Output	4	Low
HBS_GNT	Host/DMA Port Grant Operation	Input	4	High
HBS_INT	Host Interrupt	Output	4	High
HBS_RQ	Host/DMA Port Operation Request	Output	4	High
HBS_S[2:0]	Host/DMA Port DMA Transfer Size	Output	4	–
HBS_WR	Host/DMA Port Operation Type	Output	4	–
HEC_ERR	HEC Error	Output	4	High
PRU_CLK	Pacing Rate Unit Clock	Input	4	–
RC_ACK	ACI Receiver Data Acknowledge	Input	4	High

(Sheet 1 of 2)

Table 15.7 (Cont.) L64360 Pin Description Summary

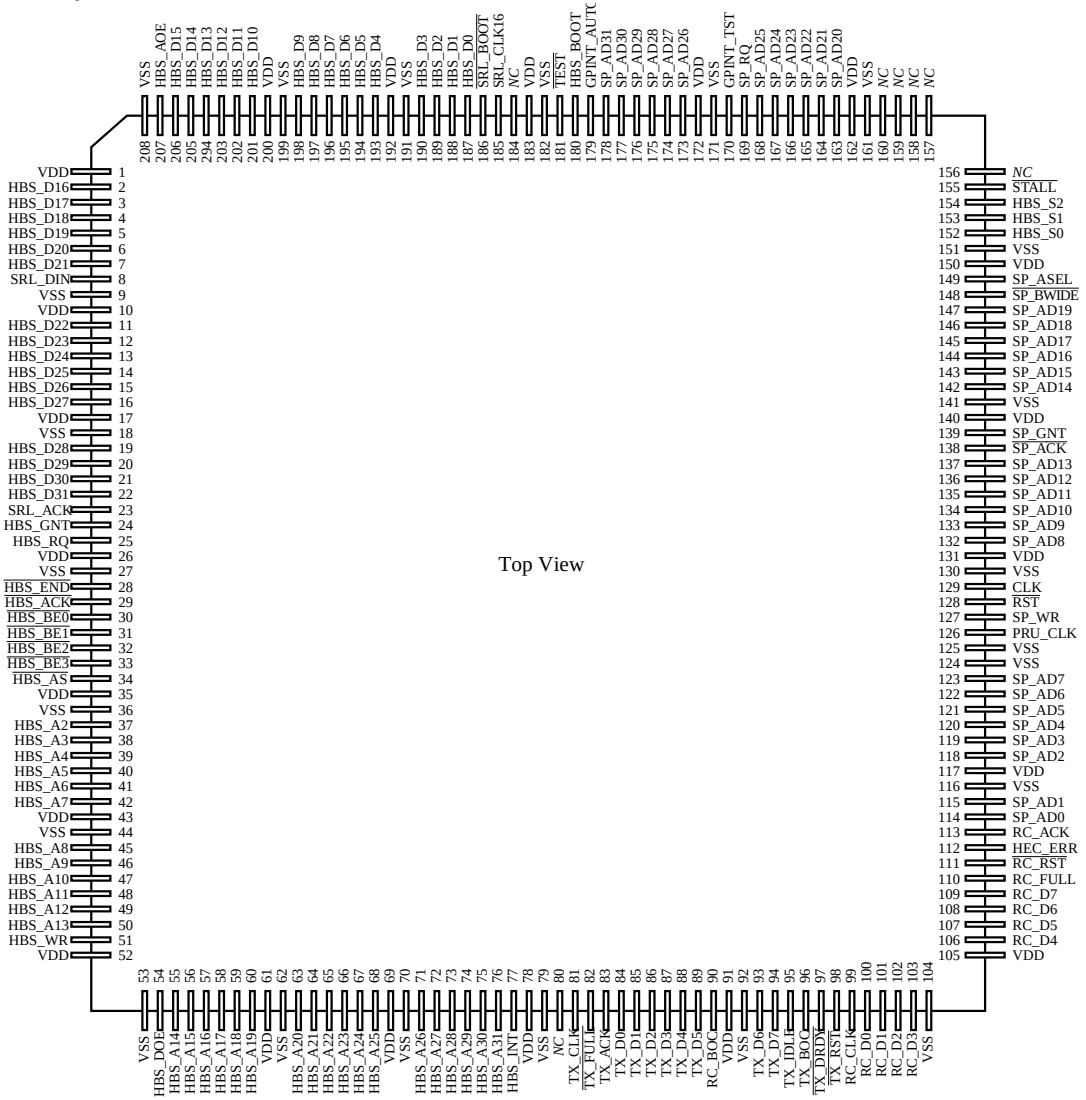
Mnemonic	*Description*	*Type*	*Drive (mA)*	*Active*
RC_BOC	ACI Receiver Beginning of Cell	Input	4	High
RC_CLK	ACI Receiver Clock	Input	4	–
RC_D[7:0]	ACI Receiver Data Bus	Input	4	–
RC_FULL	ACI Receiver Cell Holder Buffer Full	Output	4	High
$\overline{\text{RC\_RST}}$	ACI Receiver Reset	Output	4	Low
$\overline{\text{RST}}$	System Reset	Input	4	Low
SP_ACK	Secondary Port Data Acknowledge	Input	4	High
SP_AD[31:0]	Secondary Port Address/Data Bus	Bidirectional	4	–
SP_ASEL	Secondary Port Address/Data Select	Input	4	High
$\overline{\text{SP\_BWIDE}}$	Secondary Port Byte-wide Device	Input	4	Low
SP_GNT	Secondary Port Bus Grant	Input	4	High
SP_RQ	Secondary Port Access Request	Output	4	High
SP_WR	Secondary Port Operation Type	Output	4	High
$\overline{\text{STALL}}$	APU Pipeline Stall	Output	4	Low
SRL_ACK	Serial Acknowledge	Input	4	High
$\overline{\text{SRL\_BOOT}}$	Serial Boot Select	Input	4	Low
SRL_CLK16	Serial Clock	Output	4	–
SRL_DIN	Serial Data Input	Input	4	–
$\overline{\text{TEST}}$	Test Mode	Input	4	Low
TX_ACK	ACI Transmitter Data Acknowledge	Input	4	High
TX_BOC	ACI Transmitter Beginning of Cell	Output	4	High
TX_CLK	ACI Transmitter Clock	Input	4	–
TX_D[7:0]	ACI Transmitter Data Bus	Output	4	–
$\overline{\text{TX\_DRDY}}$	ACI Transmitter Data Ready	Output	4	Low
$\overline{\text{TX\_FULL}}$	ACI Transmitter Buffer Full	Input	4	Low
TX_IDLE	ACI Transmitter Idle Cell	Output	4	High
$\overline{\text{TX\_RST}}$	ACI Transmitter Reset	Output	4	Low

(Sheet 2 of 2)

15.4 Pinout, Pin List, and Package Information

The LSI Logic L64360 ASSP is available in a 208-pin, cavity down, metal quad flat package (MQUAD). Figure 15.10 illustrates the pinout of the 208-pin package, and Figure 15.11 is the mechanical drawing. Table 15.8 shows the L64360 pin list.

Figure 15.10
208-Pin MQUAD Pinout
– Cavity Down



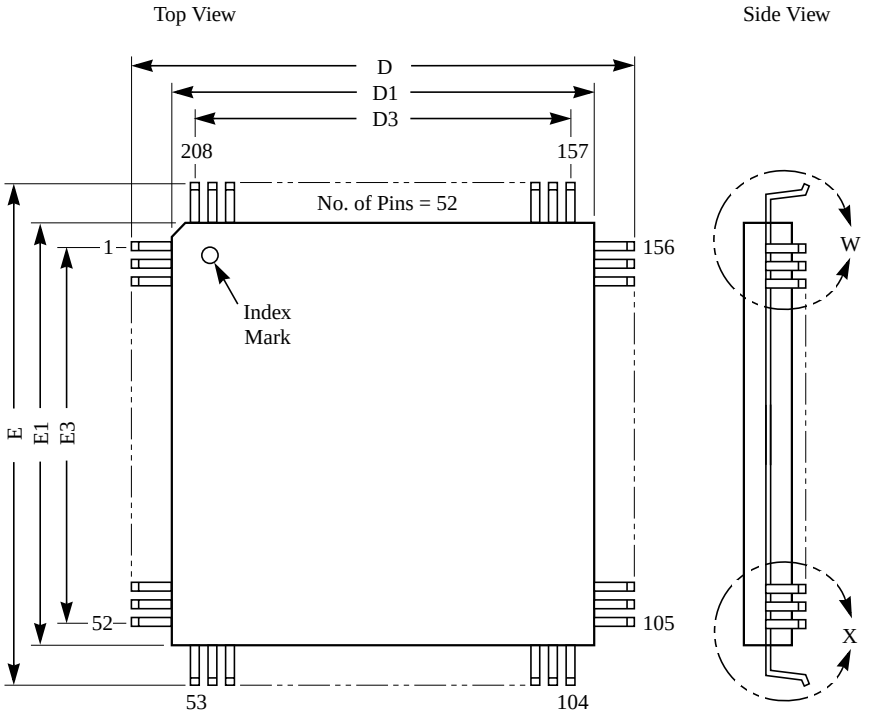
Note:
1. *NC* pins are not connected.

Table 15.8
L64360 Pin List

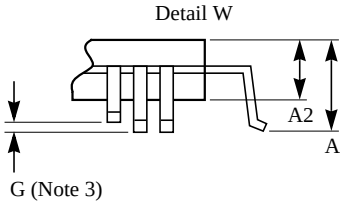
Signal	***Pin***	***Signal***	***Pin***	***Signal***	***Pin***	***Signal***	***Pin***	***Signal***	***Pin***
CLK	129	HBS_D30	21	*NC*	156	SP_AD11	135	VDD	35
GPINT_AUTO	179	HBS_D29	20	*NC*	157	SP_AD10	134	VDD	43
GPINT_TST	170	HBS_D28	19	*NC*	158	SP_AD9	133	VDD	52
HBS_A31	76	HBS_D27	16	*NC*	159	SP_AD8	132	VDD	61
HBS_A30	75	HBS_D26	15	*NC*	160	SP_AD7	123	VDD	69
HBS_A29	74	HBS_D25	14	*NC*	184	SP_AD6	122	VDD	78
HBS_A28	73	HBS_D24	13	PRU_CLK	126	SP_AD5	121	VDD	91
HBS_A27	72	HBS_D23	12	RC_ACK	113	SP_AD4	120	VDD	105
HBS_A26	71	HBS_D22	11	RC_BOC	90	SP_AD3	119	VDD	117
HBS_A25	68	HBS_D21	7	RC_CLK	99	SP_AD2	118	VDD	131
HBS_A24	67	HBS_D20	6	RC_D7	109	SP_AD1	115	VDD	140
HBS_A23	66	HBS_D19	5	RC_D6	108	SP_AD0	114	VDD	150
HBS_A22	65	HBS_D18	4	RC_D5	107	SP_ASEL	149	VDD	162
HBS_A21	64	HBS_D17	3	RC_D4	106	$\overline{\text{SP\_BWIDE}}$	148	VDD	172
HBS_A20	63	HBS_D16	2	RC_D3	103	SP_GNT	139	VDD	183
HBS_A19	60	HBS_D15	206	RC_D2	102	SP_RQ	169	VDD	192
HBS_A18	59	HBS_D14	205	RC_D1	101	SP_WR	127	VDD	200
HBS_A17	58	HBS_D13	204	RC_D0	100	SRL_ACK	23	VSS	9
HBS_A16	57	HBS_D12	203	RC_FULL	110	$\overline{\text{SRL\_BOOT}}$	186	VSS	18
HBS_A15	56	HBS_D11	202	$\overline{\text{RC\_RST}}$	111	SRL_CLK16	185	VSS	27
HBS_A14	55	HBS_D10	201	$\overline{\text{RST}}$	128	SRL_DIN	8	VSS	36
HBS_A13	50	HBS_D9	198	$\overline{\text{SP\_ACK}}$	138	$\overline{\text{STALL}}$	155	VSS	44
HBS_A12	49	HBS_D8	197	SP_AD31	178	$\overline{\text{TEST}}$	181	VSS	53
HBS_A11	48	HBS_D7	196	SP_AD30	177	TX_ACK	83	VSS	62
HBS_A10	47	HBS_D6	195	SP_AD29	176	TX_BOC	96	VSS	70
HBS_A9	46	HBS_D5	194	SP_AD28	175	TX_CLK	81	VSS	79
HBS_A8	45	HBS_D4	193	SP_AD27	174	TX_D7	94	VSS	92
HBS_A7	42	HBS_D3	190	SP_AD26	173	TX_D6	93	VSS	104
HBS_A6	41	HBS_D2	189	SP_AD25	168	TX_D5	89	VSS	116
HBS_A5	40	HBS_D1	188	SP_AD24	167	TX_D4	88	VSS	124
HBS_A4	39	HBS_D0	187	SP_AD23	166	TX_D3	87	VSS	125
HBS_A3	38	$\overline{\text{HBS\_DOE}}$	54	SP_AD22	165	TX_D2	86	VSS	130
HBS_A2	37	$\overline{\text{HBS\_END}}$	28	SP_AD21	164	TX_D1	85	VSS	141
$\overline{\text{HBS\_ACK}}$	29	HBS_GNT	24	SP_AD20	163	TX_D0	84	VSS	151
HBS_AOE	207	HBS_INT	77	SP_AD19	147	$\overline{\text{TX\_DRDY}}$	97	VSS	161
$\overline{\text{HBS\_AS}}$	34	HBS_RQ	25	SP_AD18	146	$\overline{\text{TX\_FULL}}$	82	VSS	171
$\overline{\text{HBS\_BE3}}$	33	HBS_S2	154	SP_AD17	145	TX_IDLE	95	VSS	182
$\overline{\text{HBS\_BE2}}$	32	HBS_S1	153	SP_AD16	144	$\overline{\text{TX\_RST}}$	98	VSS	191
$\overline{\text{HBS\_BE1}}$	31	HBS_S0	152	SP_AD15	143	VDD	1	VSS	199
$\overline{\text{HBS\_BE0}}$	30	HBS_WR	51	SP_AD14	142	VDD	10	VSS	208
HBS_BOOT	180	HEC_ERR	112	SP_AD13	137	VDD	17		
HBS_D31	22	*NC*[1]	80	SP_AD12	136	VDD	26		

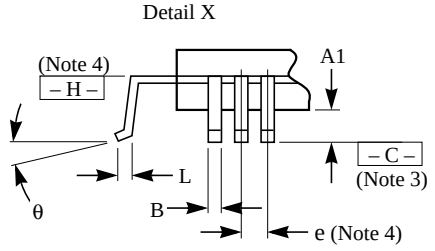
1. NC pins are not connected.

Figure 15.11
208-Pin MQUAD
Mechanical Drawing



Dimension		mm
A	**Max**	3.86
A1	**Min**	0.25
	Max	0.51
A2	**Min**	3.17
	Max	3.43
B	**Typ**	0.23
D	**Min**	30.4
	Max	30.8
D1	**Min**	27.5
	Max	27.7
D	**BSC**	25.5
E	**Min**	30.4
	Max	30.8
E1	**Min**	27.5
	Max	27.7
E	**BSC**	25.5
θ	**Min**	0°
	Max	7°
e	**BSC**	0.50
G	**Max**	0.10
L	**Min**	0.40
	Max	0.60





Note:
1. Total number of pins is 208.
2. Drawing is not to scale.
3. Coplanarity of all leads shall be within 0.10 mm (difference between the highest and lowest lead with seating plane – C – as reference).
4. Lead pitch determined at – H –.
5. Leadframe to package offset tolerance is ±0.10 mm Max.
6. For board layout and manufacturing, you may obtain engineering drawings from your LSI Logic Products marketing representative by requesting the outline drawing for package code

MD92.WVa

Appendix A
Glossary of Abbreviations

Table A.1 defines some of the abbreviations used in this manual.

Table A.1 Abbreviation Glossary

Abbreviation	***Meaning***
ASSP	Application Specific Standard Product
ATM	Asynchronous Transfer Mode
APU	ATMizer Processing Unit
AAL	ATM Adaptation Layer
ACI	ATMizer Cell Interface
BIU	Bus Interface Unit
BOC	Beginning of Cell
BOM	Beginning of Message
COM	Continuation of Message
CGCR	Channel Group Credit Register
CLP	Cell Loss Priority
CRC	Cycle (or Cyclic) Redundancy Check
CPE	Channel Parameter Entry
CSIC	Customer Specific Integrated Circuit
CS-PDU	Convergence Sublayer-Protocol Data Unit
CS-SDU	Convergence Sublayer-Service Data Unit
DRDY	Data Ready
DMA	Direct Memory Access
DMAC	DMA Controller
EOM	End of Message
FIFO	First In-First Out
GPRR	Global Pacing Rate Register
HEC	Header Error Check
IRAM	Instruction RAM

(Sheet 1 of 2)

Table A.1 (Cont.) Abbreviation Glossary

Abbreviation	***Meaning***
LO	Local Offset (VCR Offset)
LAR	Local Address Register
NIC	Network Interface Card
OAM	Operation and Management
PDU	Protocol Data Unit
PRU	Pacing Rate Unit
PRPC	Peak Rate Pacing Counter
PTI	Payload Type Indicator
RTS	Real Time Stamp
SAR	Segmentation and Reassembly
SAR-PDU	SAR-Protocol Data Units
SAR-SDU	SAR-Service Data Unit
TCS	Transmission Convergence Sublayer

(Sheet 2 of 2)

Appendix B
Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at:

LSI Logic Corporation
Microprocessor Publications
M/S G-712
Fax 408.433.8989

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

Please fax your comments to:

LSI Logic Corporation
Microprocessor Publications
M/S G-712
Fax 408.433.8989

Does the publication meet your needs? Yes__ No__

Is the material:

- Complete? Yes__ No__
- Easy to use? Yes__ No__
- Written for the appropriate technical level? Yes__ No__

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.

Name _____ Date _____
Telephone _____ Fax _____
Title _____
Department _____ Mail Stop _____
Company Name _____
Street _____
City _____
State _____ Zip _____

Your feedback is important to us. Thank you for your comments.

U.S. Distributors by State

Alabama
Huntsville
Hamilton Hallmark
Tel: 205.837.8700

Wyle Laboratories
Tel: 205.830.1119

Arizona
Phoenix
Hamilton Hallmark
Tel: 602.437.1200

Wyle Laboratories
Tel: 602.437.2088

California
Calabassas
♦ Wyle Laboratories
Tel: 818.880.9000

Costa Mesa
♦ Hamilton Hallmark
Tel: 714.641.4100

Irvine
♦ Wyle Laboratories
Tel: 714.863.9953

Sacramento
Hamilton Hallmark
Tel: 916.624.9781

Wyle Laboratories
Tel: 916.638.5282

San Diego
Hamilton Hallmark
Tel: 619.571.7540

Woodland Hills
Hamilton Hallmark
Tel: 818.594.0404

Wyle Laboratories
Tel: 619.565.9171

Santa Clara
♦ Wyle Laboratories
Tel: 408.727.2500

Sunnyvale
♦ Hamilton Hallmark
Tel: 408.435.3500

Colorado
Denver
Hamilton Hallmark
Tel: 303.790.1662

♦ Wyle Laboratories
Tel: 303.457.9953

Connecticut
Danbury
Hamilton Hallmark
Tel: 203.271.2844

Florida
Deerfield Beach
Wyle Laboratories
Tel: 305.420.0500

Miami/Ft Lauderdale
Hamilton Hallmark
Tel: 305.484.5482

St Petersburg
Hamilton Hallmark
Tel: 813.541.7440

Georgia
Atlanta
Hamilton Hallmark
Tel: 404.623.5475

Norcross
Wyle Laboratories
Tel: 404.441.9043

Illinois
Chicago
♦ Hamilton Hallmark
Tel: 708.860.7780

Addison
Wyle Laboratories
Tel: 708.303.1040

Indiana
Indianapolis
Hamilton Hallmark
Tel: 317.872.8875

Kansas
Kansas City
Hamilton Hallmark
Tel: 913.888.4747

Maryland
Baltimore
♦ Hamilton Hallmark
Tel: 401.988.9800

Columbia
Wyle Laboratories
Tel: 301.490.2170

Massachusetts
Boston
♦ Hamilton Hallmark
Tel: 508.532.9893

♦ Wyle Laboratories
Tel: 617.272.7300

Michigan
Detroit
Hamilton Hallmark
Tel: 313.347.4271

Minnesota
Minneapolis
Hamilton Hallmark
Tel: 612.881.2600

Bloomington
Wyle Laboratories
Tel: 612.853.2280

Missouri
St. Louis
Hamilton Hallmark
Tel: 314.291.5350

New Jersey
North Jersey
Hamilton Hallmark
Tel: 201.515.1641

Pine Brook
Wyle Laboratories
Tel: 201.882.8358

South Jersey
Hamilton Hallmark
Tel: 609.424.0110

New Mexico
Alburquerque
Hamilton Hallmark
Tel: 505.828.1058

New York
Long Island
Hamilton Hallmark
Tel: 516.737.0600

Rochester
Hamilton Hallmark
Tel: 716.475.9130

North Carolina
Raleigh
Hamilton Hallmark
Tel: 919.872.0712

Ohio
Cleveland
Hamilton Hallmark
Tel: 216.498.1100

Dayton
Hamilton Hallmark
Tel: 614.888.3313

Oklahoma
Tulsa
Hamilton Hallmark
Tel: 216.498.1100

Oregon
Portland
Hamilton Hallmark
Tel: 503.526.6200

Wyle Laboratories
Tel: 503.643.7900

Texas
Austin
Hamilton Hallmark
Tel: 512.258.8848

Wyle Laboratories
Tel: 512.345.8853

Dallas
Hamilton Hallmark
Tel: 214.553.4300

Wyle Laboratories
Tel: 214.235.9953

Houston
Hamilton Hallmark
Tel: 713.781.6100

Wyle Laboratories
Tel: 713.879.9953

Utah
Salt Lake City
Hamilton Hallmark
Tel: 801.266.2022

Wyle Laboratories
Tel: 801.974.9953

Washington
Seattle
Hamilton Hallmark
Tel: 206.881.6697

Wyle Laboratories
Tel: 206.881.1150

Wisconsin
Milwaukee
Hamilton Hallmark
Tel: 414.780.7200

Waukesha
Wyle Laboratories
Tel: 414.521.9333

♦ Distributors with Design Resource Centers

Sales Offices and Design Resource Centers

LSI Logic Corporation
U.S. Headquarters
Milpitas CA
♦ **Tel: 408.433.8000**
Fax: 408.433.8989

California
♦ Tel: 714.553.5600
Fax: 714.474.8101

San Diego
♦ Tel: 619.635.1300
Fax: 619.635.1350

Florida
Tel: 407.728.9481
Fax: 407.728.9587

Boca Raton
♦ Tel: 407.395.6200
Fax: 407.394.2865

Georgia
Tel: 404.395.3800
Fax: 404.395.3811

Illinois
♦ Tel: 708.995.1600
Fax: 708.995.1622

Maryland
♦ Tel: 301.897.5800
Fax: 301.897.8389

Columbia
Tel: 410.740.9191
Fax: 410.740.5587

Massachusetts
♦ Tel: 617.890.0180
Fax: 617.890.6158

Minnesota
♦ Tel: 612.921.8300
Fax: 612.921.8399

New Jersey
♦ Tel: 908.549.4500
Fax: 908.549.4802

New York
Tel: 914.226.1620
Fax: 914.226.1315

Victor
Tel: 716.223.8820
Fax: 716.223.8822

North Carolina
♦ Tel: 919.783.8833
Fax: 919.783.8909

Oregon
♦ Tel: 503.645.9882
Fax: 503.645.6612

Texas
Tel: 512.388.7294
Fax: 512.388.4171

Dallas
♦ Tel: 214.788.2966
Fax: 214.233.9234

Washington
♦ Tel: 206.822.4384
Fax: 206.827.2884

Australia
Reptechnic Ltd
Tel: 61.2.953.9844
Fax: 61.2.953.9683

LSI Logic Corporation of Canada Inc
Headquarters
Calgary
♦ Tel: 403.262.9292
Fax: 403.262.9494

Etobicoke
♦ Tel: 416.620.7400
Fax: 416.620.5005

Kanata
♦ Tel: 613.592.1263
Fax: 613.592.3253

Pointe Claire
♦ Tel: 514.694.2417
Fax: 514.694.2699

Denmark
EV Johanssen Electronic AS
♦ Tel: 45.31.839022
Fax: 45.31.839222

Finland
Komdell-Bexab Finland OY
Tel: 358.0.5023200
Fax: 358.0.5023294

France
Immeuble Europa
Paris
♦ Tel: 33.1.34631313
Fax: 33.1.34631319

Microel SA
Tel: 33.1.69070824
Fax: 33.1.69071723

Germany
LSI Logic GmbH
Munich
Tel: 49.89.45836.0
Fax: 49.89.45836.109

Munich
♦ Tel: 49.89.45833.0
Fax: 49.89.45833.108

Stuttgart
♦ Tel: 49.711.139690
Fax: 49.711.8661428

India
DCM Microelectronics Ltd
Tel: 91.11.642.9486
Fax: 91.11.644.1572

Israel
LSI Logic Limited
♦ Tel: 972.3.5403741
Fax: 972.3.5403747

Italy
LSI Logic SPA
♦ Tel: 39.39.6056881
Fax: 39.39.6057867

Esco Italiana SPA
Tel: 39.2.2409241
Fax: 39.2.2409255

Japan
LSI Logic KK
Headquarters
Tokyo
♦ Tel: 81.3.5463.7811
Fax: 81.3.5463.7825

Osaka
♦ Tel: 81.6.947.5281
Fax: 81.6.947.5287

LSI Logic Corporation of Korea Limited
♦ Tel: 82.2.561.2921
Fax: 82.2.554.9327

Norway
Bexab Norge AS
Tel: 47.63.833800
Fax: 47.63.832007

Singapore
Desner Electronics Pte Ltd
Tel: 65.285.1566
Fax: 65.284.9466

Eltee Electronics Pte Ltd
Tel: 65.283.0888
Fax: 65.289.5327

Spain
LSI Logic SA
Tel: 34.1.3672200
Fax: 34.1.3673151

Sweden
LSI Logic AB
Tel: 46.8.7034680
Fax: 46.8.7506647

Bexab Sweden AB
♦ Tel: 46.8.6308800
Fax: 46.8.7327058

Switzerland
LSI Logic Sulzer AG
♦ Tel: 41.32.536363
Fax: 41.32.536367

Taiwan
LSI Logic Asia-Pacific Regional Office
Taipei
♦ Tel: 886.2.755.3433
Fax: 886.2.755.5176

Jeilin Technology Corp
Tel: 886.2.248.4828
Fax: 886.2.248.9765

United Kingdom
LSI Logic Europe plc
European Headquarters
England
♦ Tel: 44.1344.426544
Fax: 44.1344.481039

Amega Electronics
Tel: 44.256.843166
Fax: 44.256.842956

LSI Logic Europe plc
Tel: 44.1753.680009
Fax: 44.1753.680179

Manhattan Skyline Ltd
Tel: 44.628.778686
Fax: 44.628.782812

♦ Sales Offices with Design Resource Centers

Printed on Recycled Paper

Printed in USA
295.5K.TP.RRD