IDT79RC32334 and IDT79RC32332 Integrated Communications Processors

(Y Revision)

RISCore™ 32300 Family

User Reference Manual

June 2002

2975 Stender Way, Santa Clara, California 95054
Telephone: (800) 345-7015 • TWX: 910-338-2070 • FAX: (408) 330-1748
Printed in U.S.A.

GENERAL DISCLAIMER

Integrated Device Technology, Inc. reserves the right to make changes to its products or specifications at any time, without notice, in order to improve design or performance and to supply the best possible product. IDT does not assume any responsibility for use of any circuitry described other than the circuitry embodied in an IDT product. The Company makes no representations that circuitry described herein is free from patent infringement or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent, patent rights or other rights, of Integrated Device Technology, Inc.

CODE DISCLAIMER

Code examples provided by IDT are for illustrative purposes only and should not be relied upon for developing applications. Any use of the code examples below is completely at your own risk. IDT MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE NONINFRINGEMENT, QUALITY, SAFETY OR SUITABILITY OF THE CODE, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. FURTHER, IDT MAKES NO REPRESENTATIONS OR WARRANTIES AS TO THE TRUTH, ACCURACY OR COMPLETENESS OF ANY STATEMENTS, INFORMATION OR MATERIALS CONCERNING CODE EXAMPLES CONTAINED IN ANY IDT PUBLICATION OR PUBLIC DISCLOSURE OR THAT IS CONTAINED ON ANY IDT INTERNET SITE. IN NO EVENT WILL IDT BE LIABLE FOR ANY DIRECT, CONSEQUENTIAL, INCIDENTAL, INDIRECT, PUNITIVE OR SPECIAL DAMAGES, HOWEVER THEY MAY ARISE, AND EVEN IF IDT HAS BEEN PREVIOUSLY ADVISED ABOUT THE POSSIBILITY OF SUCH DAMAGES. The code examples also may be subject to United States export control laws and may be subject to the export or import laws of other countries and it is your responsibility to comply with any applicable laws or regulations.

LIFE SUPPORT POLICY

Integrated Device Technology's products are not authorized for use as critical components in life support devices or systems unless a specific written agreement pertaining to such intended use is executed between the manufacturer and an officer of IDT.

1. Life support devices or systems are devices or systems which (a) are intended for surgical implant into the body or (b) support or sustain life and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any components of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

About This Manual

Notes

Introduction

This user reference manual includes hardware and software information on the RC32334 (Y revision)[1], a high performance integrated processor that combines a high performance 32-bit CPU core with system logic to provide direct connection to boot memory, main memory, I/O, and PCI. It also includes on-chip peripherals such as DMA channels, reset circuitry, interrupts, timers, and UARTs.

This is also the user reference manual for the RC32332 (Y revision) integrated processor. The information herein generally refers explicitly only to the RC32334 but is applicable to the RC32332 unless noted otherwise. Differences between the RC32334 and the RC32332 are identified in Appendix G.

Additional Information

Information not included in this manual such as mechanicals, package pin-outs, and electrical characteristics can be found in the data sheet for this device, which is available from the IDT website **(www.idt.com)** as well as through your local IDT sales representative.

Content Summary

Chapter 1, "RC32334 Device Overview," provides a complete introduction to the performance capabilities of the RC32334. Included in this chapter is a summary of features for the device as well as a system block diagram and internal register maps.

Chapter 2, "RC32300 CPU Core," describes the features of the RC32300 CPU core.

Chapter 3, "CPU Instruction Set Overview," presents a general overview on the three CPU instruction formats as well as the computational instructions of the MIPS architecture. Instruction set summary tables are also provided.

Chapter 4, "CPU Pipeline Architecture," discusses pipeline features as well as interlock and exception handling of the device's RISCore™ 32300.

Chapter 5, "Memory Management," contains a discussion on the virtual-to-physical address translation technique, TLB management, and operation modes for the RC32334. Register formats and field description tables are also provided in this chapter.

Chapter 6, "CPU Exception Processing," defines and describes the various exception types and handling processes for the RC32334. Also provided in this chapter are the CP0 register formats, their field descriptions, and general exception handling flowcharts.

Chapter 7, "Cache Organization, Operation, and Coherency," includes a general discussion on the operation of cache as well as the more specific cache attributes of the RC32334. Flowcharts and various diagrams are provided to clarify the concepts discussed in this chapter.

Chapter 8, "RC32334 Internal Bus," presents a general overview of the RC32334's internal bus that provides a connection to internal peripherals and controllers.

Chapter 9, "External Local Bus Interface," presents a general overview of the RC32334's system bus that provides an easy connection to main memory and to peripherals.

Chapter 10, "Memory Controller," provides a functional overview on the CPU core, DMA or PCI bridge generated transactions. A block diagram, register maps, signal description table, and timing diagrams for various read and write operations are also included.

1. For information on an earlier user manual that covers the Z revision, contact your IDT sales representative.

Notes

Chapter 11, "Synchronous DRAM Controller," contains a discussion on the operations and support provided by the RC32334's 32-bit SDRAM controller. Timing diagrams are provided to illustrate the different read and write transactions.

Chapter 12, "PCI Interface Controller," contains descriptions of the PCI host/satellite modes and master/target operations supported in the RC32334. Register maps and register field definitions are included.

Chapter 13, "DMA Controllers," includes descriptions on the four general purpose DMA channels and the transfer operations supported. Byte swapping between big- and little-endian is also discussed and includes examples.

Chapter 14, "Expansion Interrupt Controller," provides a functional and operational overview on this controller. This chapter includes a block diagram, signal definitions and register mapping tables for each of the 14 groups supported.

Chapter 15, "Programmable I/O (PIO) Controller," provides the signal descriptions, register mapping and programming information on the software programmable options of the RC32334's 15 peripheral pins.

Chapter 16, "Timer Controller," provides a user overview on the functions of the RC32334's nine on-chip timers. A block diagram, signal definitions and register maps are included.

Chapter 17, "UART Controller," describes the operation of the two 16550 compatible UARTs available on the RC32334. Register maps and descriptions are included.

Chapter 18, "Serial Peripheral Interface," describes the properties and operations of this interface to low-cost serial peripherals.

Chapter 19, "Clocking, Reset, and Initialization," provides a description of the clock signals that are used on the RC32334 processor and includes a discussion on the basic system clocks and system timing parameters. This chapter also provides a brief explanation on the power reduction modes for this device and a description of the RC32334 initialization and reset registers.

Chapter 20, "JTAG Boundary Scan," introduces the standard JTAG interface used for board-level debugging. A description on the Test Access Port (TAP) interface and TAP controller state assignments is also included.

Chapter 21, "EJTAG (In-circuit Emulator) Interface," describes the Debug Support Unit (DSU). It covers the debug instructions added to the MIPS II ISA instruction set as well as support functions and registers for debugging.

Appendix A, "RC32300 CPU Core Enhancements to MIPS II ISA," discusses in detail architectural enhancements to the MIPS II ISA.

Appendix B, "Opcode Map," provides an opcode map.

Appendix C, "The Timing of Cache Operations," provides a table for primary data cache operations and a table for primary instruction cache operations, as well as caveats about cache operations.

Appendix D, "RC32334/RC32332 Standby Mode Operation," discusses power reduction, in particular, the "Wait" instruction and the standby mode that follows this instruction.

Appendix E, "Coprocessor 0 Hazards," identifies the RC32334 CP0 hazards.

Appendix F, "Integer Multiply Scheduling," discusses integer multiply performance, defines instructions, and summarizes integer multiply and divide performance.

Appendix G, "RC32332 Differences," identifies the differences between the RC32334 and the RC32332.

Notes

Revision History

November 15, 2000: Initial publication.

February 5, 2001: In Chapter 12, separated PCI CPU Memory and I/O Space 1 Base Register section into two sections, one dealing with CPU Memory and the other with CPU I/O, and changed bit description to reflect CPU I/O Base uses [23:20] instead of [31:28].

February 26, 2001: Changed alternate function for uart_tx[0] from PIO[3] to PIO[1] in Table 1.2 and G.4. In Chapter 15, clarified that timer_tc_n[0] is not present in the RC32332 and added a reference in the Signal Definitions section to Tables G.2 and G.3. In Appendix G, added two tables (G.2 and G.3) to highlight the differences in PIO pin name assignments between the RC32334 and RC32332.

April 2, 2001: Made the following changes in Chapter 18: added system clock formula under Serial Peripheral Clock Register section; removed “active” from description for bit 2 in table 18.4; changed SPSE register to SPSR register in Table 18.6; in Master Programming Example, item 1, changed formula in parentheses to 3.7 MHz (67/ [(8+1) * 2]); in Master Programming Example, item 2, changed formula in parentheses to 3.7 / 2 = 1.85 MHz.

May 17, 2001: Table 17.6, “Interrupt Identity Register Fields and Descriptions,” has been revised to show that for bits 3:1 (Current Interrupt field) the value 111 has the same status and priority level as the value 011. Also, in Table 11.2, under SDRAM Organization column, 2nd category from the bottom, the data now reads “2 Mb x 16 x 4 banks” instead of “4 Mb x 16 x 4 banks.” Finally, in Table 11.6, for bit 28 (SDRAM Bank Size field), the value descriptions now omit any reference to 16M-bit and 64M-bit. These references were confusing because the RC32334 and RC32332 devices also support 128M-bit SDRAMs.

July 26, 2001: In Chapter 10, the bit address for mem_addr[25:2] was changed from 40000 to 3C00000 in Figures 10.6 through 10.30.

June 4, 2002: Made the following changes based on the introduction of Y silicon: Chapter 8, Internal Bus—changes in bit 7 description in Table 8.12, changes in Table 8.13. Chapter 11, SDRAM Controller—added more SDRAM address multiplexing and control registers (SDRAM Secondary Control), changes to Tables 11.1 and 11.2, changes in SDRAM Initialization section. Chapter 12, PCI Interface—added CPU to PCI and PCI to CPU mapping diagrams, new Memory/IO Space Base register and PCI Memory/IO Base Address registers, Target FIFOs are 16 words deep, added PCI Target Control Register and New Feature sections, added additional fields in PCI Arbitration Register (Table 12.15), added 2 new base address registers, revised Tables 12.1, 12.7, and 12.12, changed Reset for System Identification Number from 00h to 01h (Table 12.24). Chapter 13, DMA Controllers—added New Feature Configuration register, added SDRAM to PCI Arbitration Algorithm field, and revised function description for interrupt_n[3] and n[4] pins in Table 13.3. PIO chapter—added New Feature Register. Clocking, Reset and Initialization chapter—revised description in first row of Table 19.1.

Notes

Table of Contents

Notes

About This Manual

1 RC32334 Device Overview

2 RC32300 CPU Core

Notes

Notes

6 CPU Exception Processing

Notes

Notes

9 External Local Bus Interface

10 Memory Controller

Notes

Notes

13 DMA Controllers

Notes

Notes

16 Timer Controller

17 UART Controller

Notes

Notes

21 EJTAG (In-circuit Emulator) Interface

Notes

Notes

Notes

List of Tables

Notes

Notes

Notes

Notes

Notes

Notes

List of Figures

Notes

Notes

Notes

Notes

Notes

Notes

RC32334 Device Overview

Notes

Foreword

In this manual, numerous references are made to the RC32334, fewer references to the RC32332. Because the RC32334 core and the RC32332 core are essentially the same device, the information in this manual applies equally to both devices except where noted in occasional notes and footnotes in various chapters. Therefore, all references to the RC32334 should be interpreted as applying also to the RC32332 except where noted. The differences between the RC32334 and RC32332 are summarized in Appendix G.

Introduction

The RC32334 is an integrated processor that combines a 32-bit MIPS instruction set architecture (ISA) CPU core with a number of on-chip peripherals, to enable direct connection to boot memory, main memory, I/O, and PCI. The RC32334 also includes system logic for DMA, reset, interrupts, timers, and UARTs. The RC32334 integrates all of the peripherals commonly associated with an embedded system to reduce board space, design time, and effort.

Block Diagram

The RC32334 block diagram is shown in Figure 1.1. The sections that follow present an operational overview of the various peripheral interfaces and controller capabilities that comprise the RC32334 system. Also included in this chapter is a full pin description table and logic diagram. More detailed explanations and user details such as register descriptions, timing diagrams and memory maps are provided in the specific chapter for that function.

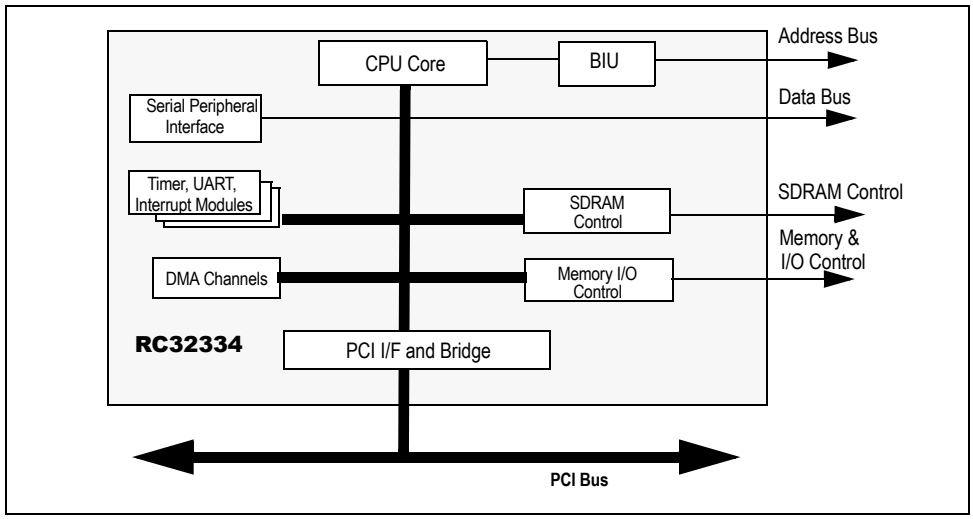


Figure 1.1 RC32334 Block Diagram

Documentation Conventions and Definitions

Note that throughout this manual the following terms and conventions will be used:

- *To avoid confusion when dealing with a mixture of "active-low" and "active-high" signals, the terms assertion and negation are used. The term assert or assertion is used to indicate that a signal is active or true, independent of whether that level is represented by a high or low voltage. The term negate or negation is used to indicate that a signal is inactive or false.*
- *To define the active polarity of a signal, a suffix will be used. Signals ending with an '_n' should be interpreted as being active, or asserted, when at a logic zero (low) level. All other signals (including*

Notes

clocks, buses and select lines) will be interpreted as being active, or asserted when at a logic one (high) level.

- *To define buses, the most significant bit (MSB) will be on the left and least significant bit (LSB) will be on the right. No leading zeros will be included.*
- *To represent numerical values, either decimal, binary, or hexadecimal formats will be used. The binary format is as follows: 0bDDD, where "D" represents either 0 or 1; the hexadecimal format is as follows: 0xDD, where "D" represents the hexadecimal digit(s); otherwise, it is decimal.*
- *Unless otherwise denoted, a byte will refer to an 8-bit quantity. A halfword will refer to a 16-bit quantity. A triple-byte will refer to a 24-bit quantity. A word will refer to a 32-bit quantity, and a double or double word will refer to a 64-bit quantity.*
- *A bit is set when its value is 0b1. A bit is cleared when its value is 0b0.*
- *The compressed notation ABC[x|y|z]D refers to ABCxD, ABCyD, and ABCzD.*
- *In words, bit 31 is always the most significant bit and bit 0 is the least significant bit. In halfwords, bit 15 is always the most significant bit and bit 0 is the least significant bit. In bytes, bit 7 is always the most significant bit and bit 0 is the least significant bit.*
- *The ordering of bytes within words is referred to as either "big endian" or "little endian." Big endian systems label byte zero as the most significant (leftmost) byte of a word. Little endian systems label byte zero as the least significant (rightmost) byte of a word.*

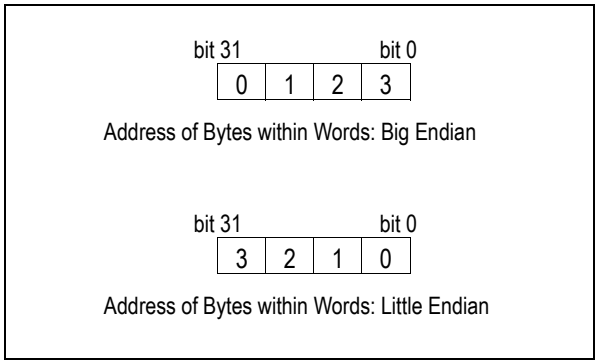


Table 1.1 Example of Byte Ordering for "Big Endian" or "Little Endian" System Definition

Signal Terminology

Throughout this manual, when describing signal transitions, the following terminology is used:

- *Rising edge indicates a low-to-high (0 to 1) transition.*
- *Falling edge indicates a high-to-low (1 to 0) transition.*
- *Clock-to-Q delay is the amount of time it takes for a signal to move from the input of a device (clock) to the output of the device (Q).*

These terms are illustrated in Figure 1.2 and Figure 1.3.

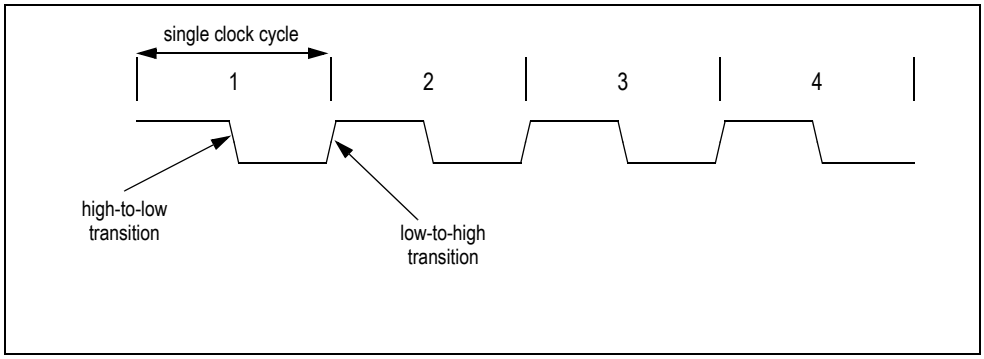


Figure 1.2 Signal Transitions

Notes

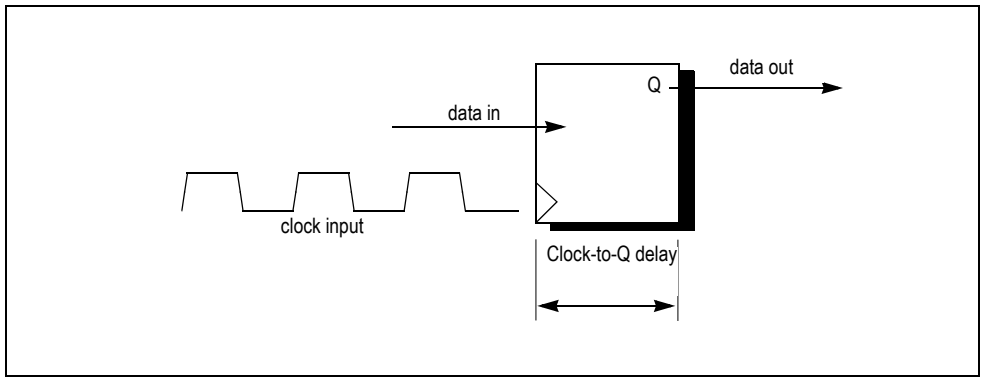


Figure 1.3 Clock-to-Q Delay

List of Features

Note: This list is not entirely applicable to the RC32332. For the differences in features between the RC32334 and RC32332, see Table G.1 in Appendix G.

- *High performance 32-bit CPU core*
  - *DSP instruction set extensions*
  - *Enhanced MIPS-II ISA compatible*
  - *8kB instruction/2kB data cache, lockable per line*
  - *Big or little endian support*
- *SDRAM Controller (32-bit memory only)*
  - *4 banks, non-interleaved, 512MB total*
  - *Automatic refresh generation*
- *Memory & Peripheral Controller*
  - *6 banks, up to 32 or 64MB per bank (bank dependent)*
  - *8/16/ or 32-bit interface per bank*
  - *Supports Flash ROM, PROM, SRAM, dual-port memory, and peripheral devices*
  - *Intel or Motorola style IO supports external wait-state generation*
- *PCI Bridge*
  - *32-bit PCI, up to 66 MHz*
  - *Revision 2.2 compliant*
  - *Target and master*
  - *Host or satellite*
  - *On-chip three slot PCI arbiter*
- *4 DMA Channels*
  - *4 general purpose DMA, each with Endianness swappers and byte lane data alignment*
  - *Any channel can be used for PCI*
  - *Supports scatter/gather*
  - *Supports memory-to-memory, memory-to-I/O, memory-to-PCI, PCI-to-PCI, and I/O-to-I/O transfers*
  - *Supports chaining via linked lists of records*
  - *Supports unaligned transfers*
  - *Supports burst transfers*
  - *Programmable DMA transactions burst size*
- *UART Interface*
  - *Two 16550 Compatible UARTs*
  - *Baud rate support up to 1.5M*
  - *Modem signals included on one channel*
- *Programmable IO (PIO)*
  - *Input/Output/Interrupt source*
  - *Individually programmable*

Notes

- *Interrupt Control*
  - *Provides services for internal and external sources*
  - *Allows status of each interrupt to be read and masked*
- *Four general purpose 32-bit timer/counters*
- *Serial Peripheral Interface (SPI)*
- *Boundary Scan JTAG Interface (IEEE Std. 1149.1 compatible)*
- *In Circuit Emulator Interface*
  - *Compatible with enhanced JTAG (EJTAG) standard*

System Block Diagram

Figure 1.4 illustrates the typical system implementation, based on the RC32334 integrated processor. The RC32334 provides all of the necessary control and address signals to drive the external memory and I/O. Note that, depending on the loading of the system bus, external data buffers could be used to reduce the loading and isolate different memory regions.

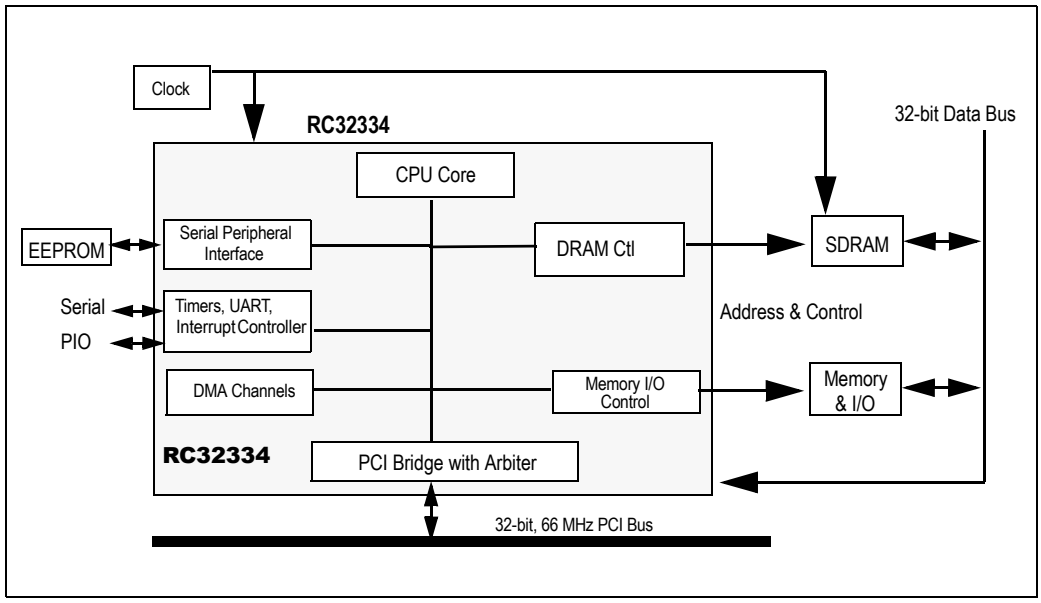


Figure 1.4 System Block Diagram

System Overview

Note: The PCI bridge information and the UART information in this section is not entirely applicable to the RC32332. For the differences, see Table G.1 in Appendix G.

The RC32334 generates all necessary control signals and address buses to the external memory and I/O. For main memory, I/O, on-chip peripherals, registers, and PCI, the RC32334 divides the physical address space into 13 different regions.

Memory Controller. The Memory Controller on the RC32334 provides all of the address buses and control signals for interfacing the RC32334 to standard SRAM, PROM, FLASH, and I/O, and includes the boot PROM interface. The memory controller provides six individual chip selects and supports 8, 16, and 32-bit wide memory and I/Os. The two chip selects have highly configurable memory address ranges, allowing selection of various memory types and widths to be supported. The RC32334 provides controls for optional external data transceivers, for systems that require fast signalling with large loads.

SDRAM Controller. The SDRAM controller provides higher throughput while using available DRAM circuitry, adding little to the cost of the system. The SDRAM controller directly manages four banks of 32-bit physical non-interleaved memory. Each bank is 32-bits wide and supports a maximum of 64 MB per bank,

Notes

up to a total memory size of 512 MB. The SDRAM memory subsystem can be implemented with a broad range of device types, including SO-Dimms and SDRAM modules with devices from 16 Mb to 256 Mb. The SDRAM controller has a built-in refresh generator.

PCI Bridge. To transfer data between main memory and the PCI bus, the RC32334 incorporates a PCI bridge. At reset time, the PCI bridge can be configured as either a host or satellite bridge. The PCI bridge supports 32-bit PCI—at up to 66MHz—and is revision 2.2 compliant.

As a PCI master, the RC32334 can generate memory, I/O, or configuration cycles for direct local-to-PCI bus accesses. The PCI bridge contains internal logic to arbitrate the ownership of the PCI bus between multiple PCI bus master devices. For up to three external PCI devices, two arbitration schemes are supported:

- *Round robin, allowing devices to control the bus in a programmable sequential order*
- *Fixed priority, allowing the user to provide more bus bandwidth to a specific PCI-based peripheral*

As a PCI target, the RC32334 allows access to its internal registers and to the RC32334 local bus through the PCI, I/O read and write, or Memory read and write commands. The RC32334 PCI bridge supports byte swapping between little endian and big endian ordering conventions for systems when the CPU subsystem is configured as a big endian system.

DMA Controller. Four general purpose DMA channels move data between source and destination ports. Source and destination ports can be system memory, PCI, or I/O devices. Any of the four channels can be used for PCI initiator reads or writes. All four channels support a descriptor structure, to allow efficient data scatter/gather. The DMA controller supports swapping of data between big and little endian memory and I/O subsystems. It also supports quad-word burst transfers. All external 16 and 8-bit memory I/Os are treated as memory-mapped, word-aligned devices.

Expansion Interrupt Controller. The Expansion Interrupt Controller provides the interrupt logic for software to analyze the various RC32334 generated system interrupts and adds to the control already provided through the CP0 registers of the RISCore™ 32300. Each system interrupt is registered and the pending status provided through this feature. The pending status can then be used to automatically generate a hardware interrupt to the CPU core via individual mask bits. The pending interrupt status can also be optionally set or cleared by a direct software write.

PIO. Programmable I/O (PIO) pins are provided on the RC32334 so that any unused peripheral pins can be programmed for use as general purpose discrete I/O pins. These PIO pins can be software programmed as input or output lines, allowing pin values to be software programmed in output mode and software readable while in the input mode. The PIO pins can also be used as a source of interrupts to the CPU.

UART. The RC32334 incorporates two 16550 (an enhanced version of the 16450) compatible UARTs. To relieve the CPU of software overhead, the 16550 UART can be put into FIFO mode, allowing execution of either 16450 or 16550 compatible software. Two sets of 16-byte FIFOs are enabled during the 16550 mode: one set in the receive data path and one set in the transmit data path. A baud rate generator is included that divides the system clock by 1 to 65K and provides a 16X clock for driving the transmitter and receiver logic.

Timers/Counters. Three on-chip 32-bit general purpose Timers are provided on the RC32334. Each timer consists of both a count and a compare register. The count register resets to zero and then increments until it equals the compare register. When the count and compare registers are equal, the TC_n output is asserted and the count is then reset to zero.

JTAG. Board-level manufacturing debugging is facilitated through implementation of a fully compliant IEEE std. 1149.1 JTAG Boundary Scan interface.

In-Circuit Emulation. The part provides an on-chip debug port, enabling an external system to access CPU core information. In conjunction with an in-circuit emulator (compatible with the EJTAG standard defined by MIPS Technologies), this enables a sophisticated system debug capability to improve the software development process.

Serial Peripheral Interface (SPI). This slow speed serial interface provides direct connection to SPI-based peripherals, including EEPROMs and analog to digital (A-to-D) converters.

Pin Description Table — RC32334

The following table lists the pins provided on the RC32334. Note that those pin names followed by "_n" are active-low signals. All external pull-ups and pull-downs require 10 kΩ resistor.

<table>
<tr><th>Name</th><th>Type</th><th>Reset State Status</th><th>Drive Strength Capability</th><th>Description</th></tr>
<tr><td colspan="5">Local System Interface</td></tr>
<tr><td>mem_data[31:0]</td><td>I/O</td><td>Z</td><td>High</td><td>Local system data bus
Primary data bus for memory. I/O and SDRAM. Requires external pull-up.</td></tr>
<tr><td>mem_addr[25:2]</td><td>I/O</td><td>[25:10] Z
[9:2] L</td><td>[25:16] Low
[15:2] High</td><td>Memory Address Bus
These signals provide the Memory or DRAM address, during a Memory or DRAM bus transaction. During each word data, the address increments either in linear or sub-block ordering, depending on the transaction type. The table below indicates how the memory write enable signals are used to address discreet memory port width types.
<table>
<tr><th>Port Width</th><th>Pin Signals
mem_we_n[3]</th><th>mem_we_n[2]</th><th>mem_we_n[1]</th><th>mem_we_n[0]</th></tr>
<tr><td>DMA (32-bit)</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr>
<tr><td>32-bit</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr>
<tr><td>16-bit</td><td>Byte High Write Enable</td><td>mem_addr[1]</td><td>Not Used (Driven Low)</td><td>Byte Low Write Enable</td></tr>
<tr><td>8-bit</td><td>Not Used (Driven High)</td><td>mem_addr[1]</td><td>mem_addr[0]</td><td>Byte Write Enable</td></tr>
</table>
mem_addr[22] Alternate function: reset_boot_mode[1].
mem_addr[21] Alternate function: reset_boot_mode[0].
mem_addr[20] Alternate function: reset_pci_host_mode.
mem_addr[19] Alternate function: modebit [9].
mem_addr[18] Alternate function: modebit [8].
mem_addr[17] Alternate function: modebit [7].
mem_addr[15] Alternate function: sdram_addr[15].
mem_addr[14] Alternate function: sdram_addr[14].
mem_addr[13] Alternate function: sdram_addr[13].
mem_addr[11] Alternate function: sdram_addr[11].
mem_addr[10] Alternate function: sdram_addr[10].
mem_addr[9] Alternate function: sdram_addr[9].
mem_addr[8] Alternate function: sdram_addr[8].
mem_addr[7] Alternate function: sdram_addr[7].
mem_addr[6] Alternate function: sdram_addr[6].
mem_addr[5] Alternate function: sdram_addr[5].
mem_addr[4] Alternate function: sdram_addr[4].
mem_addr[3] Alternate function: sdram_addr[3].
mem_addr[2] Alternate function: sdram_addr[2].</td></tr>
<tr><td>mem_cs_n[5:0]</td><td>Output</td><td>H</td><td>Low with internal pull-up</td><td>Memory Chip Select Negated
Recommend external pull-up.
Signals that a Memory Bank is actively selected.</td></tr>
<tr><td>mem_oe_n</td><td>Output</td><td>H</td><td>High</td><td>Memory Output Enable Negated
Recommend external pull-up.
Signals that a Memory Bank can output its data lines onto the cpu_ad bus.</td></tr>
<tr><td>mem_we_n[3:0]</td><td>Output</td><td>H</td><td>High</td><td>Memory Write Enable Negated Bus
Signals which bytes are to be written during a memory transaction. Bits act as Byte Enable and mem_addr[1:0] signals for 8-bit or 16-bit wide addressing.</td></tr>
</table>

Table 1.2 Pin Description for RC32334 (Part 1 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
mem_wait_n	Input		—	**Memory Wait Negated** Requires external pull-up. SRAM/IOI/IOM modes: Allows external wait-states to be injected during last cycle before data is sampled. DPM (dual-port) mode: Allows dual-port busy signal to restart memory transaction. Alternate function: sdram_wait_n.
mem_245_oe_n	Output	H	Low	**Memory FCT245 Output Enable Negated** Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to a memory or I/O bank.
mem_245_dt_r_n	Output	Z	High	**Memory FCT245 Direction Xmit/Rcv Negated** Recommend external pull-up. Alternate function: cpu_dt_r_n. See CPU Core Specific Signals below.
output_clk	Output	cpu-masterclk	High	**Output Clock** Optional clock output.

PCI Interface

pci_ad[31:0]	I/O	Z	PCI	**PCI Multiplexed Address/Data Bus** Address driven by Bus Master during initial frame_n assertion, and then the Data is driven by the Bus Master during writes; or the Data is driven by the Bus Slave during reads.
pci_cbe_n[3:0]	I/O	Z	PCI	**PCI Multiplexed Command/Byte Enable Bus** Command (not negated) Bus driven by the Bus Master during the initial frame_n assertion. Byte Enable Negated Bus driven by the Bus Master during the data phase(s).
pci_par	I/O	Z	PCI	**PCI Parity** Even parity of the pci_ad[31:0] bus. Driven by Bus Master during Address and Write Data phases. Driven by the Bus Slave during the Read Data phase.
pci_frame_n	I/O	Z	PCI	**PCI Frame Negated** Driven by the Bus Master. Assertion indicates the beginning of a bus transaction. De-assertion indicates the last datum.
pci_trdy_n	I/O	Z	PCI	**PCI Target Ready Negated** Driven by the Bus Slave to indicate the current datum can complete.
pci_irdy_n	I/O	Z	PCI	**PCI Initiator Ready Negated** Driven by the Bus Master to indicate that the current datum can complete.
pci_stop_n	I/O	Z	PCI	**PCI Stop Negated** Driven by the Bus Slave to terminate the current bus transaction.
pci_idsel_n	Input		—	**PCI Initialization Device Select** Uses pci_req_n[2] pin. See the PCI subsection.
pci_perr_n	I/O	Z	PCI	**PCI Parity Error Negated** Driven by the receiving Bus Agent 2 clocks after the data is received, if a parity error occurs.
pci_serr_n	I/O Open-collector	Z	PCI	**System Error** External pull-up resistor is required. Driven by any agent to indicate an address parity error, data parity during a Special Cycle command, or any other system error.
pci_clk	Input		—	**PCI Clock** Clock for PCI Bus transactions. Uses the rising edge for all timing references.
pci_rst_n	Input	L	—	**PCI Reset Negated** Host mode: Resets all PCI related logic. Satellite mode: with boot from PCI mode: Resets all PCI related logic and also warm resets the 32334.
pci_devsel_n	I/O	Z	PCI	**PCI Device Select Negated** Driven by the target to indicate that the target has decoded the present address as a target address.

Table 1.2 Pin Description for RC32334 (Part 2 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
pci_req_n[2]	Input	Z	—	**PCI Bus Request #2 Negated** Requires external pull-up. Host mode: pci_req_n[2] is an input indicating a request from an external device. Satellite mode: used as pci_idsel pin which selects this device during a configuration read or write. Alternate function: pci_idsel (satellite).
pci_req_n[1]	Input	Z	—	**PCI Bus Request #1 Negated** Requires external pull-up. Host mode: pci_req_n[2] is an input indicating a request from an external device. Alternate function: Unused (satellite).
pci_req_n[0]	I/O	Z	High	**PCI Bus Request #0 Negated** Requires external pull-up for burst mode. Host mode: pci_req_n[0] is an input indicating a request from an external device. Satellite mode: pci_req_n[0] is an output indicating a request from this device.
pci_gnt_n[2]	Output	Z[1]	High	**PCI Bus Grant #2 Negated** Recommend external pull-up. Host mode: pci_gnt_n[2] is an output indicating a grant to an external device. Satellite mode: pci_gnt_n[2] is used as the pci_inta_n output pin. Alternate function: pci_inta_n (satellite).
pci_gnt_n[1] / pci_eeprom_cs	I/O	X for 1 pci clock then H[2]	High	**PCI Bus Grant #1 Negated** Recommend external pull-up. Host mode: pci_gnt_n[2:1] are outputs indicating grants to external devices. Satellite mode: Used as pci_eprom_cs output pin for Serial Chip Select for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: pci_eeprom_cs (satellite). 2nd Alternate function: PIO[11].
pci_gnt_n[0]	I/O	Z	High	**PCI Bus Grant #0 Negated** Host mode: pci_gnt_n[0] is an output indicating a grant to an external device. Recommend external pull-up. Satellite mode: pci_gnt_n[0] is an input indicating a grant to this device. Require external pull-up.
pci_inta_n	Output Open-collector	Z	PCI	**PCI Interrupt #A Negated** Uses pci_gnt_n[2]. See the PCI subsection.
pci_lock_n	Input		—	**PCI Lock Negated** Driven by the Bus Master to indicate that an exclusive operation is occurring.

[1] Z in host mode; L in satellite non-boot mode; Z in satellite boot mode.
[2] H in host mode; L in satellite non-boot mode; L in satellite boot mode.

SDRAM Control Interface

sdram_addr_12	Output	L	High	**SDRAM Address Bit 12 and Precharge All** SDRAM mode: Provides SDRAM address bit 12 (10 on the SDRAM chip) during row address and "precharge all" signal during refresh, read and write command.
sdram_ras_n	Output	H	High	**SDRAM RAS Negated** SDRAM mode: Provides SDRAM RAS control signal to all SDRAM banks.
sdram_cas_n	Output	H	High	**SDRAM CAS Negated** SDRAM mode: Provides SDRAM CAS control signal to all SDRAM banks.
sdram_we_n	Output	H	High	**SDRAM WE Negated** SDRAM mode: Provides SDRAM WE control signal to all SDRAM banks.
sdram_cke	Output	H	High	**SDRAM Clock Enable** SDRAM mode: Provides clock enable to all SDRAM banks.

Table 1.2 Pin Description for RC32334 (Part 3 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
sdram_cs_n[3:0]	Output	H	High	**SDRAM Chip Select Negated Bus** Recommend external pull-up. SDRAM mode: Provides chip select to each SDRAM bank. SODIMM mode: Provides upper select byte enables [7:4].
sdram_s_n[1:0]	Output	H	High	**SDRAM SODIMM Select Negated Bus** SDRAM mode: Not used. SDRAM SODIMM mode: Upper and lower chip selects.
sdram_bemask_n [3:0]	Output	H	High	**SDRAM Byte Enable Mask Negated Bus (DQM)** SDRAM mode: Provides byte enables for each byte lane of all DRAM banks. SODIMM mode: Provides lower select byte enables [3:0].
sdram_245_oe_n	Output	H	Low	**SDRAM FCT245 Output Enable Negated** Recommend external pull-up. SDRAM mode: Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to any DRAM bank.
sdram_245_dt_r_n	Output	Z	High	**SDRAM FCT245 Direction Transmit/Receive** Recommend external pull-up. Uses cpu_dt_r_n. See CPU Core Specific Signals below.
On-Chip Peripherals				
dma_ready_n[1:0] / dma_done_n[1:0]	I/O	Z	Low	**DMA Ready Negated Bus** Requires external pull-up. Ready mode: Input pin for each general purpose DMA channel that can initiate the next datum in the current DMA descriptor frame. Done mode: Input pin for each general purpose DMA channel that can terminate the current DMA descriptor frame. dma_ready_n[0] 1st Alternate function PIO[1]; 2nd Alternate function: dma_done_n[0]. dma_ready_n[1] 1st Alternate function PIO[0]; 2nd Alternate function: dma_done_n[1].
pio[15:0]	I/O	See related pins	Low	**Programmable Input/Output** General purpose pins that can each be configured as a general purpose input or general purpose output. These pins are multiplexed with other pin functions: uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], uart_rts_n[0], pci_gnt_n[1], spi_mosi, spi_miso, spi_sck, spi_ss_n, uart_rx[0], uart_tx[0], uart_rx[1], uart_tx[1], timer_tc_n[0], dma_ready_n[0], dma_ready_n[1]. Note that pci_gnt_n[1], spi_mosi, spi_sck, and spi_ss_n default to outputs at reset time. The others default to inputs.
timer_tc_n[0] / timer_gate_n[0]	I/O	Z	Low	**Timer Terminal Count Overflow Negated** Terminal count mode (timer_tc_n): Output indicating that the timer has reached its count compare value and has overflowed back to 0. Gate mode (timer_gate_n): input indicating that the timer may count one tick on the next clock edge. 1st Alternate function: PIO[2]. 2nd Alternate function: timer_gate_n[0].
uart_rx[1:0]	I/O	Z	Low	**UART Receive Data Bus** UART mode: Each UART channel receives data on their respective input pin. uart_rx[0] Alternate function: PIO[6]. uart_rx[1] Alternate function: PIO[4].
uart_tx[1:0]	I/O	Z	Low	**UART Transmit Data Bus** UART mode: Each UART channel sends data on their respective output pin. Note that these pins default to inputs at reset time and must be programmed via the PIO interface before being used as UART outputs. uart_tx[0] Alternate function: PIO[5]. uart_tx[1] Alternate function: PIO[3].

Table 1.2 Pin Description for RC32334 (Part 4 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
uart_cts_n[0] uart_dsr_n[0] uart_dtr_n[0] uart_rts_n[0]	I/O	Z	Low	**UART Transmit Data Bus** UART mode: Data bus modem control signal pins for UART channel 0. uart_cts_n[0] Alternate function: PIO[15]. uart_dsr_n[0] Alternate function: PIO[14]. uart_dtr_n[0] Alternate function: PIO[13]. uart_rts_n[0] Alternate function: PIO[12].
spi_mosi	I/O	L	Low	**SPI Data Output** Serial mode: Output pin from RC32334 as an Input to a Serial Chip for the Serial data input stream. In PCI satellite mode, acts as an Output pin from RC32334 that connects as an Input to a Serial Chip for the Serial data input stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. 1st Alternate function: PIO[10]. Defaults to the output direction at reset time. 2nd Alternate function: pci_eeprom_mdo.
spi_miso	I/O	Z	Low	**SPI Data Input** Serial mode: Input pin to RC32334 from the Output of a Serial Chip for the Serial data output stream. In PCI satellite mode, acts as an Input pin from RC32334 that connects as an output to a Serial Chip for the Serial data output stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Defaults to input direction at reset time. 1st Alternate function: PIO[7]. 2nd Alternate function: pci_eeprom_mdi.
spi_sck	I/O	L	Low	**SPI Clock** Serial mode: Output pin for Serial Clock. In PCI satellite mode, acts as an Output pin for Serial Clock for loading PCI Configuration Registers in the RC323334 Reset Initialization Vector PCI boot mode. 1st Alternate function: PIO[9]. Defaults to the output direction at reset time. 2nd Alternate function: pci_eeprom_sk.
spi_ss_n	I/O	H	Low	**SPI Chip Select** Output pin selecting the serial protocol device as opposed to the PCI satellite mode EEPROM device. Alternate function: PIO[8]. Defaults to the output direction at reset time.

CPU Core Specific Signals

cpu_nmi_n	Input		—	**CPU Non-Maskable Interrupt** Requires external pull-up. This interrupt input is active low to the CPU.
cpu_masterclk	Input		—	**CPU Master System Clock** Provides the basic system clock.
cpu_int_n[5:4], [2:0]	Input		—	**CPU Interrupt** Requires external pull-up. These interrupt inputs are active low to the CPU.
cpu_coldreset_n	Input	L	—	**CPU Cold Reset** This active-low signal is asserted to the RC32334 after V_{cc} becomes valid on the initial power-up. The Reset initialization vectors for the RC32334 are latched by cold reset.
cpu_dt_r_n	Output	Z	—	**CPU Direction Transmit/Receive** This active-low signal controls the DT/R pin of an optional FCT245 transceiver bank. It is asserted during read operations. Requires external pull-up. 1st Alternate function: mem_245_dt_r_n. 2nd Alternate function: sdram_245_dt_r_n.

Table 1.2 Pin Description for RC32334 (Part 5 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
JTAG Interface Signals				
jtag_tck	Input		—	**JTAG Test Clock** Requires external pull-down. An input test clock used to shift into or out of the Boundary-Scan register cells. jtag_tck is independent of the system and the processor clock with nominal 50% duty cycle.
jtag_tdi, ejtag_dint_n	Input		—	**JTAG Test Data In** Requires an external pull-up on the board. On the rising edge of jtag_tck, serial input data are shifted into either the Instruction or Data register, depending on the TAP controller state. During Real Mode, this input is used as an interrupt line to stop the debug unit from Real Time mode and return the debug unit back to Run Time Mode (standard JTAG). This pin is also used as the ejtag_dint_n signal in the EJTAG mode.
jtag_tdo, ejtag_tpc	Output	Z	High	**JTAG Test Data Out** The jtag_tdo is serial data shifted out from instruction or data register on the falling edge of jtag_tck. When no data is shifted out, the jtag_tdo is tri-stated. During Real Time Mode, this signal provides a non-sequential program counter at the processor clock or at a division of processor clock. This pin is also used as the ejtag_tpc signal in the EJTAG mode.
jtag_tms	Input		—	**JTAG Test Mode Select** Requires external pull-up. The logic signal received at the jtag_tms input is decoded by the TAP controller to control test operation. jtag_tms is sampled on the rising edge of the jtag_tck.
jtag_trst_n	Input	L	—	**JTAG Test Reset** When neither JTAG nor EJTAG are being used, jtag_trst_n must be driven or pulled low, or the jtag_tms/ ejtag_tms signals must be pulled up and jtag_clk actively clocked.
ejtag_dclk	Output	Z	—	**EJTAG Test Clock** Processor Clock. During Real Time Mode, this signal is used to capture address and data from the ejtag_tpc signal at the processor clock speed or any division of the internal pipeline.
ejtag_pcst[2:0]	I/O	Z	Low	**EJTAG PC Trace Status Information** 111 (STL) Pipe line Stall 110 (JMP) Branch/Jump forms with PC output 101 (BRT) Branch/Jump forms with no PC output 100 (EXP) Exception generated with an exception vector code output 011 (SEQ) Sequential performance 010 (TST) Trace is outputted at pipeline stall time 001 (TSQ) Trace trigger output at performance time 000 (DBM) Run Debug Mode Alternate function: modebit[2:0].
ejtag_debugboot	Input		— Requires external pull-down	**EJTAG DebugBoot** The ejtag_debugboot input is used during reset and forces the CPU core to take a debug exception at the end of the reset sequence instead of a reset exception. This enables the CPU to boot from the ICE probe without having the external memory working. This input signal is level sensitive and is not latched internally. This signal will also set the JtagBrk bit in the JTAG_Control_Register[12].
ejtag_tms	Input		— Requires external pull-up	**EJTAG Test Mode Select** An external pull-up on the board is required. The ejtag_tms is sampled on the rising edge of jtag_tck.
Debug Signals				
debug_cpu_dma_n	I/O	Z	Low	**Debug CPU versus DMA Negated** De-assertion high during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from the CPU. Assertion low during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from DMA. Alternate function: modebit[6].

Table 1.2 Pin Description for RC32334 (Part 6 of 7)

Name	Type	Reset State Status	Drive Strength Capability	Description
debug_cpu_ack_n	I/O	Z	Low	**Debug CPU Acknowledge Negated** Indicates either a data acknowledge to the CPU or DMA. Alternate function: modebit[4].
debug_cpu_ads_n	I/O	Z	Low	**Debug CPU Address/Data Strobe Negated** Assertion indicates that either a CPU or a DMA transaction is beginning and that the mem_data[31:4] bus has the current block address. Alternate function: modebit[5].
debug_cpu_i_d_n	I/O	Z	Low	**Debug CPU Instruction versus Data Negated** Assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU or DMA data transaction. De-assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU instruction transaction. Alternate function: modebit[3].

Table 1.2 Pin Description for RC32334 (Part 7 of 7)

Pin Description Table — RC32332

The following table lists the pins provided on the RC32332. Note that those pin names followed by "_n" are active-low signals. All external pull-ups and pull-downs require 10 kΩ resistor.

Name	Type	Reset State Status	Drive Strength Capability	Description
Local System Interface				
mem_data[31:0]	I/O	Z	High	**Local system data bus** Primary data bus for memory. I/O and SDRAM.
mem_addr[22:2]	I/O	[25:10] Z [9:2] L	[22:16] Low [15:2] High	**Memory Address Bus** These signals provide the Memory or DRAM address, during a Memory or DRAM bus transaction. During each word data, the address increments either in linear or sub-block ordering, depending on the transaction type. The table below indicates how the memory write enable signals are used to address discrete memory port width types. mem_addr[22] Alternate function: reset_boot_mode[1]. mem_addr[21] Alternate function: reset_boot_mode[0]. mem_addr[20] Alternate function: reset_pci_host_mode. mem_addr[19] Alternate function: modebit [9]. mem_addr[18] Alternate function: modebit [8]. mem_addr[17] Alternate function: modebit [7]. mem_addr[15] Alternate function: sdram_addr[15]. mem_addr[14] Alternate function: sdram_addr[14]. mem_addr[13] Alternate function: sdram_addr[13]. mem_addr[11] Alternate function: sdram_addr[11]. mem_addr[10] Alternate function: sdram_addr[10]. mem_addr[9] Alternate function: sdram_addr[9]. mem_addr[8] Alternate function: sdram_addr[8]. mem_addr[7] Alternate function: sdram_addr[7]. mem_addr[6] Alternate function: sdram_addr[6]. mem_addr[5] Alternate function: sdram_addr[5]. mem_addr[4] Alternate function: sdram_addr[4]. mem_addr[3] Alternate function: sdram_addr[3]. mem_addr[2] Alternate function: sdram_addr[2]
mem_cs_n[5:0]	Output	H	Low	**Memory Chip Select Negated** Recommend external pull-up. Signals that a Memory Bank is actively selected.
mem_oe_n	Output	H	High	**Memory Output Enable Negated** Recommend external pull-up. Signals that a Memory Bank can output its data lines onto the cpu_ad bus.
mem_we_n[3:0]	Output	H	High	**Memory Write Enable Negated Bus** Signals which bytes are to be written during a memory transaction. Bits act as Byte Enable and mem_addr[1:0] signals for 8-bit or 16-bit wide addressing.
mem_wait_n	Input		—	**Memory Wait Negated** Requires external pull-up. SRAM/IOI/IOM modes: Allows external wait-states to be injected during the last cycle before data is sampled. DPM (dual-port) mode: Allows dual-port busy signal to restart memory transaction. Alternate function: sdram_wait_n.

Port Width	Pin Signals mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
DMA (32-bit)	mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
32-bit	mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
16-bit	Byte High Write Enable	mem_addr[1]	Not Used (Driven Low)	Byte Low Write Enable
8-bit	Not Used (Driven High)	mem_addr[1]	mem_addr[0]	Byte Write Enable

Table 1.3 Pin Description for RC32332 (Part 1 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
mem_245_oe_n	Output	H	Low	**Memory FCT245 Output Enable Negated** Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to a memory or I/O bank.
mem_245_dt_r_n	Output	Z	High	**Memory FCT245 Direction Xmit/Rcv Negated** Recommend external pull-up. Alternate function: cpu_dt_r_n. See CPU Core Specific Signals below.
output_clk	Output	cpu-masterclk	High	**Output Clock** Optional clock output.
PCI Interface				
pci_ad[31:0]	I/O	Z	PCI	**PCI Multiplexed Address/Data Bus** Address driven by Bus Master during initial frame_n assertion, and then the Data is driven by the Bus Master during writes; or the Data is driven by the Bus Slave during reads.
pci_cbe_n[3:0]	I/O	Z	PCI	**PCI Multiplexed Command/Byte Enable Bus** Command (not negated) Bus driven by the Bus Master during the initial frame_n assertion. Byte Enable Negated Bus driven by the Bus Master during the data phase(s).
pci_par	I/O	Z	PCI	**PCI Parity** Even parity of the pci_ad[31:0] bus. Driven by Bus Master during Address and Write Data phases. Driven by the Bus Slave during the Read Data phase.
pci_frame_n	I/O	Z	PCI	**PCI Frame Negated** Driven by the Bus Master. Assertion indicates the beginning of a bus transaction. De-assertion indicates the last datum.
pci_trdy_n	I/O	Z	PCI	**PCI Target Ready Negated** Driven by the Bus Slave to indicate the current datum can complete.
pci_irdy_n	I/O	Z	PCI	**PCI Initiator Ready Negated** Driven by the Bus Master to indicate that the current datum can complete.
pci_stop_n	I/O	Z	PCI	**PCI Stop Negated** Driven by the Bus Slave to terminate the current bus transaction.
pci_idsel_n	Input		—	**PCI Initialization Device Select** Uses pci_req_n[2] pin. See the PCI subsection.
pci_perr_n	I/O	Z	PCI	**PCI Parity Error Negated** Driven by the receiving Bus Agent 2 clocks after the data is received, if a parity error occurs.
pci_serr_n	I/O Open-collector	Z	PCI	**System Error** External pull-up resistor is required. Driven by any agent to indicate an address parity error, data parity during a Special Cycle command, or any other system error.
pci_clk	Input		—	**PCI Clock** Clock for PCI Bus transactions. Uses the rising edge for all timing references.
pci_rst_n	Input	L	—	**PCI Reset Negated** Host mode: Resets all PCI related logic. Satellite mode: with boot from PCI mode: Resets all PCI related logic and also warm resets the 32332.
pci_devsel_n	I/O	Z	PCI	**PCI Device Select Negated** Driven by the target to indicate that the target has decoded the present address as a target address.
pci_req_n[2]	Input	Z	—	**PCI Bus Request #2 Negated** Requires external pull-up. Host mode: pci_req_n[2] is an input indicating a request from an external device. Satellite mode: used as pci_idsel pin which selects this device during a configuration read or write. Alternate function: pci_idsel (satellite).
pci_req_n[0]	I/O	Z	High	**PCI Bus Request #0 Negated** Requires external pull-up for burst mode. Host mode: pci_req_n[0] is an input indicating a request from an external device. satellite mode: pci_req_n[0] is an output indicating a request from this device.

Table 1.3 Pin Description for RC32332 (Part 2 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
pci_gnt_n[2]	Output	Z[1]	High	**PCI Bus Grant #2 Negated** Recommend external pull-up. Host mode: pci_gnt_n[2] is an output indicating a grant to an external device. Satellite mode: pci_gnt_n[2] is used as the pci_inta_n output pin. External pull-up is required. Alternate function: pci_inta_n (satellite).
pci_gnt_n[1]	I/O	X for 1 pci clock then H[2]	High	**PCI Bus Grant #1 Negated** Recommend external pull-up. Host mode: not used. Satellite mode: Used as pci_eprom_cs output pin for Serial Chip Select for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: pci_eeprom_cs (satellite). 2nd Alternate function: PIO[7].
pci_gnt_n[0]	I/O	Z	High	**PCI Bus Grant #0 Negated** Host mode: pci_gnt_n[0] is an output indicating a grant to an external device. Recommend external pull-up. Satellite mode: pci_gnt_n[0] is an input indicating a grant to this device. Requires external pull-up.
pci_inta_n	Output Open-collector	Z	PCI	**PCI Interrupt #A Negated** Uses pci_gnt_n[2]. See the PCI subsection.
pci_lock_n	Input		—	**PCI Lock Negated** Driven by the Bus Master to indicate that an exclusive operation is occurring.
[1] Z in host mode; L in satellite non-boot mode; Z in satellite boot mode. [2] H in host mode; L in satellite non-boot mode; L in satellite boot mode.				
SDRAM Control Interface				
sdram_addr_12	Output	L	High	**SDRAM Address Bit 12 and Precharge All** SDRAM mode: Provides SDRAM address bit 12 (10 on the SDRAM chip) during row address and “precharge all” signal during refresh, read and write command.
sdram_ras_n	Output	H	High	**SDRAM RAS Negated** SDRAM mode: Provides SDRAM RAS control signal to all SDRAM banks.
sdram_cas_n	Output	H	High	**SDRAM CAS Negated** SDRAM mode: Provides SDRAM CAS control signal to all SDRAM banks.
sdram_we_n	Output	H	High	**SDRAM WE Negated** SDRAM mode: Provides SDRAM WE control signal to all SDRAM banks.
sdram_cke	Output	H	High	**SDRAM Clock Enable** SDRAM mode: Provides clock enable to all SDRAM banks.
sdram_cs_n[3:0]	Output	H	High	**SDRAM Chip Select Negated Bus** Recommend external pull-up. SDRAM mode: Provides chip select to each SDRAM bank. SODIMM mode: Provides upper select byte enables [7:4].
sdram_s_n[1:0]	Output	H	High	**SDRAM SODIMM Select Negated Bus** SDRAM mode: Not used. SDRAM SODIMM mode: Upper and lower chip selects.
sdram_bemask_n [3:0]	Output	H	High	**SDRAM Byte Enable Mask Negated Bus (DQM)** SDRAM mode: Provides byte enables for each byte lane of all DRAM banks. SODIMM mode: Provides lower select byte enables [3:0].

Table 1.3 Pin Description for RC32332 (Part 3 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
sdram_245_oe_n	Output	H	Low	**SDRAM FCT245 Output Enable Negated** Recommend external pull-up. SDRAM mode: Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to any DRAM bank.
sdram_245_dt_r_n	Output	Z	High	**SDRAM FCT245 Direction Transmit/Receive** Recommend external pull-up. Uses cpu_dt_r_n. See CPU Core Specific Signals below.
On-Chip Peripherals				
dma_ready_n[0]	I/O	Z	Low	**DMA Ready Negated Bus** Requires external pull-up. Ready mode: Input pin for general purpose DMA channel 0 that can initiate the next datum in the current DMA descriptor frame. Done mode: Input pin for general purpose DMA channel 0 that can terminate the current DMA descriptor frame. dma_ready_n[0] 1st Alternate function PIO[0]; 2nd Alternate function: dma_done_n[0].
pio[7:0]	I/O	See related pins	Low	**Programmable Input/Output** General purpose pins that can each can be configured as a general purpose input or general purpose output. These pins are multiplexed with other pin functions: pci_gnt_n[1], spi_mosi, spi_miso, spi_sck, spi_ss_n, uart_rx[0], uart_tx[0], dma_ready_n[0]. Note that pci_gnt_n[1], spi_mosi, spi_sck, and spi_ss_n default to outputs at reset time. The others default to inputs.
uart_rx[0]	I/O	Z	Low	**UART Receive Data Bus** UART mode: UART channel receives data. uart_rx[0] Alternate function: PIO[2].
uart_tx[0]	I/O	Z	Low	**UART Transmit Data** UART mode: UART channel send data. Note that this pin defaults to an input at reset time and must be programmed via the PIO interface before being used as a UART output. uart_tx[0] Alternate function: PIO[1].
spi_mosi	I/O	L	Low	**SPI Data Output** Serial mode: Output pin from RC32332 as an Input to a Serial Chip for the Serial data input stream. In PCI satellite mode, acts as an Output pin from RC32332 that connects as an Input to a Serial Chip for the Serial data input stream for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[6]. 2nd Alternate function: pci_eeprom_mdo.
spi_miso	I/O	Z	Low	**SPI Data Input** Serial mode: Input pin to RC32332 from the Output of a Serial Chip for the Serial data output stream. In PCI satellite mode, acts as an Input pin from RC32332 that connects as an output to a Serial Chip for the Serial data output stream for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. 1st Alternate function: PIO[3]. 2nd Alternate function: pci_eeprom_mdi.
spi_sck	I/O	L	Low	**SPI Clock** Serial mode: Output pin for Serial Clock. In PCI satellite mode, acts as an Output pin for Serial Clock for loading PCI Configuration Registers in the RC323332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[5]. 2nd Alternate function: pci_eeprom_sk.
spi_ss_n	I/O	H	Low	**SPI Chip Select** Output pin selecting the serial protocol device as opposed to the PCI satellite mode EEPROM device. Alternate function: PIO[4]. Defaults to the output direction at reset time.

Table 1.3 Pin Description for RC32332 (Part 4 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
CPU Core Specific Signals				
cpu_nmi_n	Input		—	**CPU Non-Maskable Interrupt** Requires external pull-up. This interrupt input is active low to the CPU.
cpu_masterclk	Input		—	**CPU Master System Clock** Provides the basic system clock.
cpu_int_n[1:0]	Input		—	**CPU Interrupt** Requires external pull-up. These interrupt inputs are active low to the CPU.
cpu_coldreset_n	Input	L	—	**CPU Cold Reset** This active-low signal is asserted to the RC32332 after V_{cc} becomes valid on the initial power-up. The Reset initialization vectors for the RC32332 are latched by cold reset.
cpu_dt_r_n	Output	Z	—	**CPU Direction Transmit/Receive** This active-low signal controls the DT/R pin of an optional FCT245 transceiver bank. It is asserted during read operations. 1st Alternate function: mem_245_dt_r_n. 2nd Alternate function: sdram_245_dt_r_n.
JTAG Interface Signals				
jtag_tck	Input		—	**JTAG Test Clock** Requires external pull-down. An input test clock used to shift into or out of the Boundary-Scan register cells. jtag_tck is independent of the system and the processor clock with nominal 50% duty cycle.
jtag_tdi, ejtag_dint_n	Input		—	**JTAG Test Data In** Requires an external pull-up on the board. On the rising edge of jtag_tck, serial input data are shifted into either the Instruction or Data register, depending on the TAP controller state. During Real Mode, this input is used as an interrupt line to stop the debug unit from Real Time mode and return the debug unit back to Run Time Mode (standard JTAG). Requires an external pull-up on the board. This pin is also used as the ejtag_dint_n signal in the EJTAG mode.
jtag_tdo, ejtag_tpc	Output	Z	High	**JTAG Test Data Out** The jtag_tdo is serial data shifted out from instruction or data register on the falling edge of jtag_tck. When no data is shifted out, the jtag_tdo is tri-stated. During Real Time Mode, this signal provides a non-sequential program counter at the processor clock or at a division of processor clock. This pin is also used as the ejtag_tpc signal in the EJTAG mode.
jtag_tms	Input		—	**JTAG Test Mode Select** Requires external pull-up. The logic signal received at the jtag_tms input is decoded by the TAP controller to control test operation. jtag_tms is sampled on the rising edge of the jtag_tck.
jtag_trst_n	Input	L	—	**JTAG Test Reset** When neither JTAG nor EJTAG are being used, jtag_trst_n must be driven or pulled low, or the jtag_tms/ ejtag_tms signals must be pulled up and jtag_clk actively clocked.
ejtag_dclk	Output	Z	—	**EJTAG Test Clock** Processor Clock. During Real Time Mode, this signal is used to capture address and data from the ejtag_tpc signal at the processor clock speed or any division of the internal pipeline.

Table 1.3 Pin Description for RC32332 (Part 5 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
ejtag_pcst[2:0]	I/O	Z	Low	**EJTAG PC Trace Status Information** 111 (STL) Pipe line Stall 110 (JMP) Branch/Jump forms with PC output 101 (BRT) Branch/Jump forms with no PC output 100 (EXP) Exception generated with an exception vector code output 011 (SEQ) Sequential performance 010 (TST) Trace is outputted at pipeline stall time 001 (TSQ) Trace trigger output at performance time 000 (DBM) Run Debug Mode Alternate function: modebit[2:0].
ejtag_debugboot	Input		—	**EJTAG DebugBoot** Requires an external pull-down. The ejtag_debugboot input is used during reset and forces the CPU core to take a debug exception at the end of the reset sequence instead of a reset exception. This enables the CPU to boot from the ICE probe without having the external memory working. This input signal is level sensitive and is not latched internally. This signal will also set the JtagBrk bit in the JTAG_Control_Register[12].
ejtag_tms	Input		—	**EJTAG Test Mode Select** Requires an external pull-up. The ejtag_tms is sampled on the rising edge of jtag_tck.
Debug Signals				
debug_cpu_dma_n	I/O	Z	Low	**Debug CPU versus DMA Negated** Assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from the CPU. De-assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from DMA. Alternate function: modebit[6].
debug_cpu_ack_n	I/O	Z	Low	**Debug CPU Acknowledge Negated** Indicates either a data acknowledge to the CPU or DMA. Alternate function: modebit[4].
debug_cpu_ads_n	I/O	Z	Low	**Debug CPU Address/Data Strobe Negated** Assertion indicates that either a CPU or a DMA transaction is beginning and that the mem_data[31:4] bus has the current block address. Alternate function: modebit[5].
debug_cpu_i_d_n	I/O	Z	Low	**Debug CPU Instruction versus Data Negated** Assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU or DMA data transaction. De-assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU instruction transaction. Alternate function: modebit[3].

Table 1.3 Pin Description for RC32332 (Part 6 of 6)

Logic Diagram — RC32334

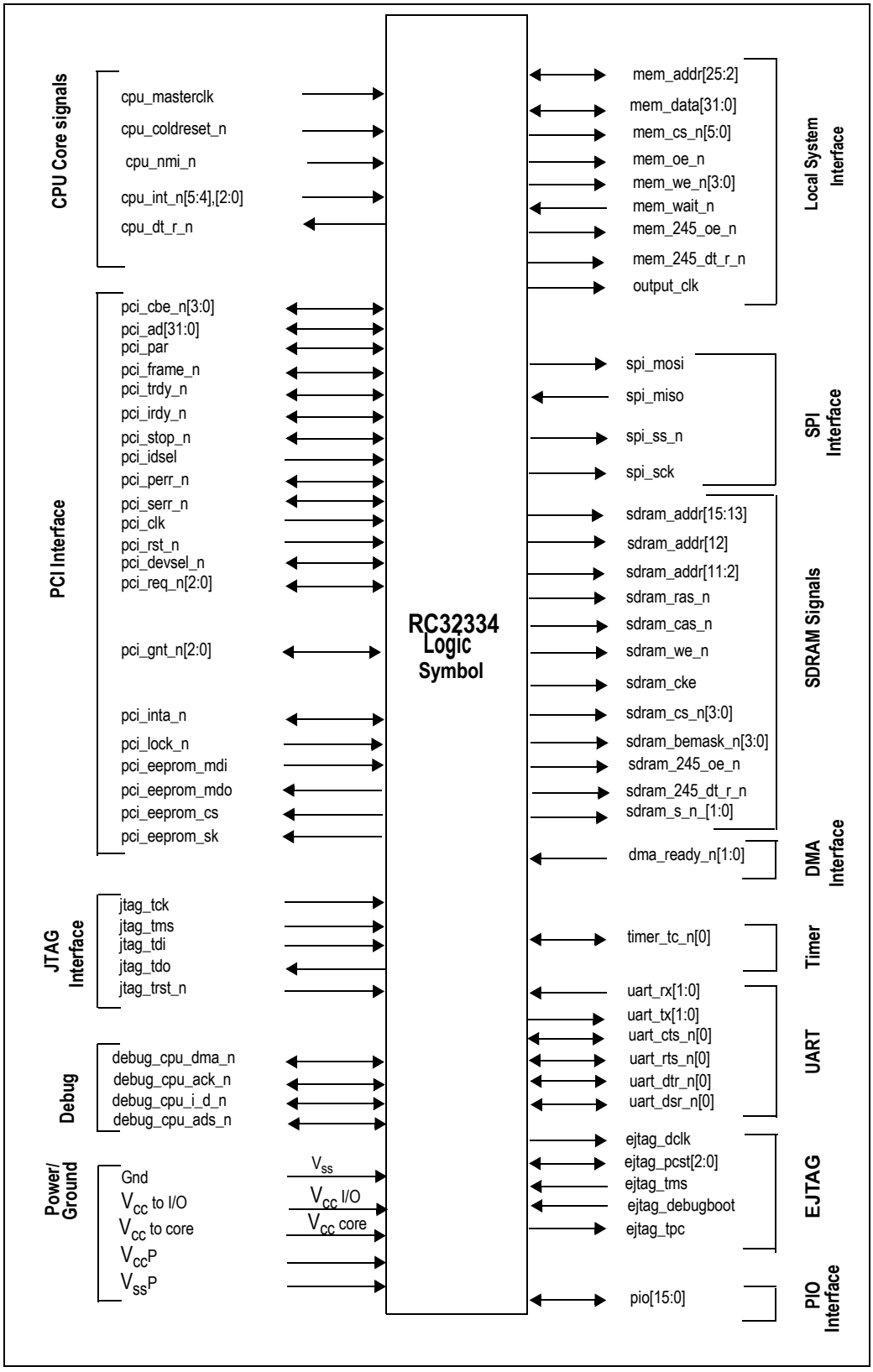


Figure 1.5 Logic Diagram for RC32334

Logic Diagram — RC32332

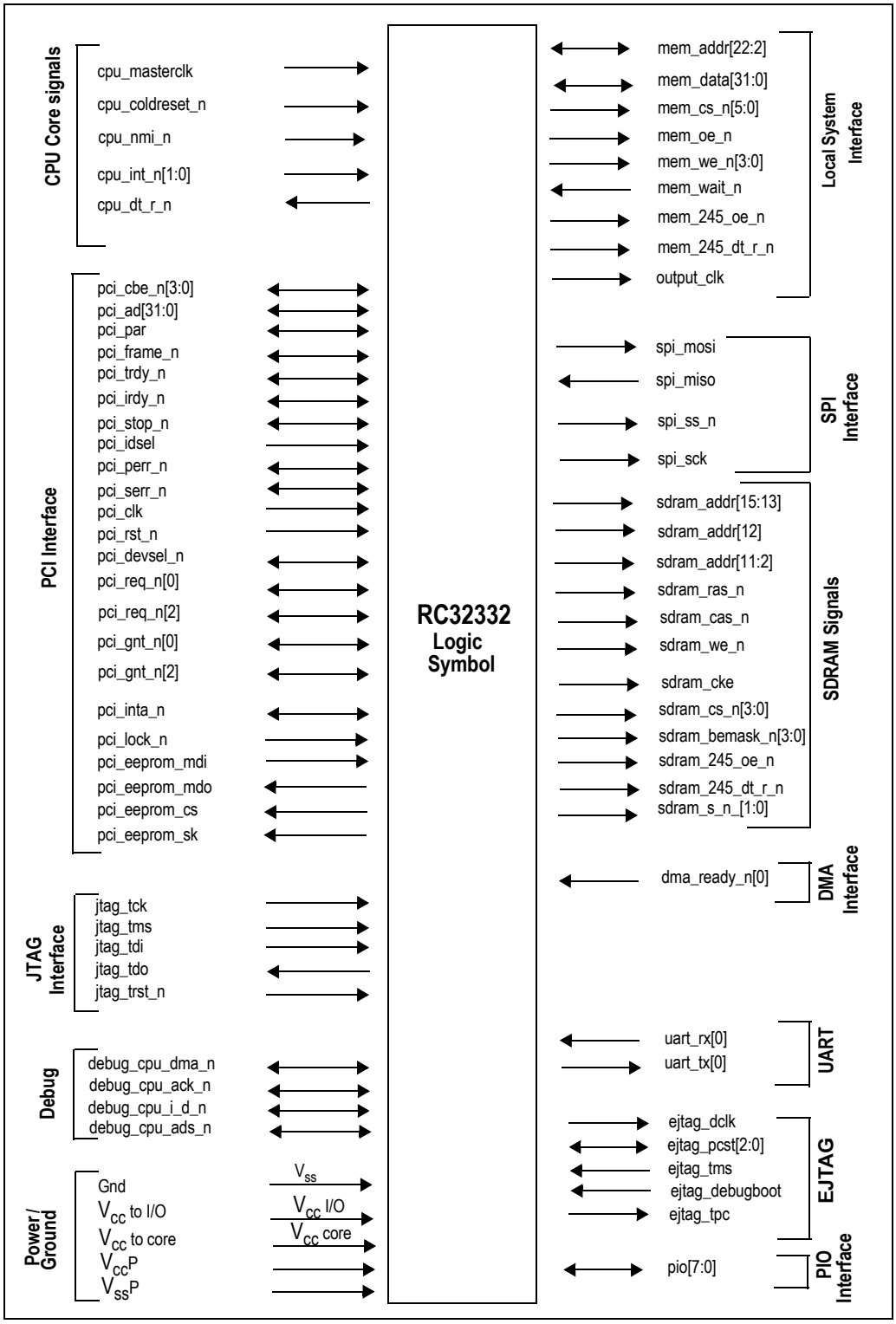


Figure 1.6 Logic Diagram for RC32332

Notes

Typical RC32334 Memory Map

The RC32334 divides the physical address range into 12 distinctive regions and decodes the address generated by the CPU to determine which region is being accessed. The RC32334 integrated processor allows rearrangement of the memory map for particular embedded applications such as systems with SRAM main memory. Typical RC32334 systems use the following memory map.

Physical Address Range		Max. Size	Port Size Region[1]	RC32334 Region Description
0000_0000	0FFF_FFFF	256MB	A,B,C,D	DRAM 0,1,2,3 (16MB typical)
1000_0000	11FF_FFFF	32MB	E	MEM/IO 2 (8MB typical)
1200_0000	13FF_FFFF	32MB	E	MEM/IO 3 (8MB typical)
1400_0000	15FF_FFFF	32MB	F	MEM/IO 4 (8MB typical)
1600_0000	17FF_FFFF	32MB	F	MEM/IO 5 (8MB typical)
1800_0000	1BFF_FFFF	64MB	G	RC32334 Internal Registers
1C00_0000	1FFF_FFFF	64MB	H	MEM/IO 0 (4MB typical)
2000_0000	23FF_FFFF	64MB	I	MEM/IO 1 (8MB typical)
4000_0000	5FFF_FFFF	512MB	J	PCI Memory Space 1 (256MB typical)
6000_0000	7FFF_FFFF	512MB	K	PCI Memory Space 2 (256MB typical)
8000_0000	FF1F_FFFF	2034MB	O	Reserved, undecoded. If accesses are made to this region, the RC32334 will return a bus error.
FF20_0000	FF2F_FFFF	1MB	O	Reserved for RC32334 EJTAG interface Probe Memory.
FF30_0000	FFFF_FFFF	13MB	O	Reserved for RC32334 on-chip EJTAG interface registers.

Table 1.4 RC32334 Typical Memory Map

[1] Multiple memory regions may need to be placed within the larger RISCore 32300 port size regions.

Note: Typical values are values a programmer might use. They are not necessarily the default values at reset.

RC32334 Internal Register Map Addresses and Definitions

Important User's Note: If required, the internal register addresses listed below can be changed by the reset initialization mode—from base 1800_0000 to base 1900_0000. Non-boot mode, PCI-boot mode, and Standard-boot mode sequence settings are provided in the Reset group of the RC32334 Pin Descriptions, Table 1.2.

This section does not include the RC32300 CPU core internal registers. For more details on the RC32300 CPU core registers, see Chapter 2, which describes the registers inside of the CPU core.

BIU Control Registers

The BIU Control registers are special interface registers used to control the bus access and bus error functions of the interface unit.

Notes

Base Address	Register Function	Offset Address	Effective Address
1800_0000	BTA Register	00	Base + Offset
	Address Latch Timing Register	04	
	Arbitration Register	08	
1800_0000	BusError Control Register	10	Base + Offset
	BusError Address Register	14	
	SysID Register	18	

Table 1.5 Internal Address Map for BIU Control Registers

Base Address and Base Mask Registers

These registers are used to select the address to be decoded for memory banks 0,1, 2 or 3. The base address registers determine the starting location of a particular memory chip select, and the mask registers are used for address bit comparison and chip select activation. Functional descriptions of these registers are provided in Chapters 9 and 11.

Base Address	Register Function	Offset Address	Effective Address
1800_0000	Memory Base Address Register for Bank 0	80	Base + Offset
	Memory Base Mask Register for Bank 0	84	
	Memory Base Address Register for Bank 1	88	
	Memory Base Mask Register for Bank 1	8C	
	DRAM Base Address Register for Bank 0	C0	
	DRAM Base Mask Register for Bank 0	C4	
	DRAM Base Address Register for Bank 1	C8	
	DRAM Base Mask Register for Bank 1	CC	
	DRAM Base Address Register for Bank 2	D0	
	DRAM Base Mask Register for Bank 2	D4	
	DRAM Base Address Register for Bank 3	D8	
	DRAM Base Mask Register for Bank 3	DC	

Table 1.6 Internal Address Map for Memory and DRAM Base Address and Base Mask Registers

Notes

Memory Control Registers

These registers provide control of the memory resource options, such as type, size, and wait-state assertion for banks 0 through 5. Register definitions and descriptions are provided in Chapter 10, Memory Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0200	Memory Control Register Bank 0	00	Base + Offset
	Memory Control Register Bank 1	04	
	Memory Control Register Bank 2	08	
	Memory Control Register Bank 3	0C	
	Memory Control Register Bank 4	10	
	Memory Control Register Bank 5	14	

Table 1.7 Internal Address Map for Memory Control Registers

DRAM Memory Controller Registers

These registers direct control of the DRAM resources. Resource specific definitions and descriptions are included in Chapter 11, Synchronous DRAM Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0300	SDRAM Control Register	00	Base + Offset
	Reserved	10	

Table 1.8 Internal Address Map for DRAM Memory Controller Registers

Expansion Interrupt Registers

These register groups manage the interrupts. Each grouping has 3 registers: interrupt pending, interrupt mask and interrupt clear. The register function is the same from group to group; however, each interrupt is group specific. Register definitions are provided in Chapter 14, Expansion Interrupt Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0500	Expansion Interrupt Pending Register 0	00	Base + Offset
	Expansion Interrupt Mask Register 0	04	
	Expansion Interrupt Clear Register 0	08	
1800_0500	Expansion Interrupt Pending Register 1	10	Base + Offset
	Expansion Interrupt Mask Register 1	14	
	Expansion Interrupt Clear Register 1	18	
1800_0500	Expansion Interrupt Pending Register 2	20	Base + Offset
	Expansion Interrupt Mask Register 2	24	
	Expansion Interrupt Clear Register 2	28	

Table 1.9 Internal Address Mapping of Expansion Interrupt Registers (Part 1 of 2)

Notes

Base Address	Register Function	Offset Address	Effective Address
1800_0500	Expansion Interrupt Pending Register 3	30	Base + Offset
	Expansion Interrupt Mask Register 3	34	
	Expansion Interrupt Clear Register 3	38	
1800_0500	Expansion Interrupt Pending Register 4	40	Base + Offset
	Expansion Interrupt Mask Register 4	44	
	Expansion Interrupt Clear Register 4	48	
1800_0500	Expansion Interrupt Pending Register 5	50	Base + Offset
	Expansion Interrupt Mask Register 5	54	
	Expansion Interrupt Clear Register 5	58	
1800_0500	Expansion Interrupt Pending Register 6	60	Base + Offset
	Expansion Interrupt Mask Register 6	64	
	Expansion Interrupt Clear Register 6	68	
1800_0500	Expansion Interrupt Pending Register 7	70	Base + Offset
	Expansion Interrupt Mask Register 7	74	
	Expansion Interrupt Clear Register 7	78	
1800_0500	Expansion Interrupt Pending Register 8	80	Base + Offset
	Expansion Interrupt Mask Register 8	84	
	Expansion Interrupt Clear Register 8	88	
1800_0500	Expansion Interrupt Pending Register 9	90	Base + Offset
	Expansion Interrupt Mask Register 9	94	
	Expansion Interrupt Clear Register 9	98	
1800_0500	Expansion Interrupt Pending Register 10	A0	Base + Offset
	Expansion Interrupt Mask Register 10	A4	
	Expansion Interrupt Clear Register 10	A8	
1800_0500	Expansion Interrupt Pending Register 11	B0	Base + Offset
	Expansion Interrupt Mask Register 11	B4	
	Expansion Interrupt Clear Register 11	B8	
1800_0500	Expansion PCI Interrupt Pending Register 12	C0	Base + Offset
	Expansion PCI Interrupt Mask Register 12	C4	
	Expansion PCI Interrupt Clear Register 12	C8	
1800_0500	Expansion Interrupt Pending Register 13	D0	Base + Offset
	Expansion Interrupt Mask Register 13	D4	
	Expansion Interrupt Clear Register 13	D8	
1800_0500	Expansion Interrupt Pending Register 14	E0	Base + Offset
	Expansion Interrupt Mask Register 14	E4	
	Expansion Interrupt Clear Register 14	E8	

Table 1.9 Internal Address Mapping of Expansion Interrupt Registers (Part 2 of 2)

Notes

Programmable I/O Registers

These registers allow I/O programmability between internal peripheral and general purpose functions. Register definitions and user operations are provided in Chapter 15, Programmable I/O (PIO) Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0600	PIO Data Register 0	00	Base + Offset
	PIO Direction Control Register 0	04	
	PIO Effect Select Control Register 0	08	
	PIO New Feature Register 0	0C	
1800_0610	PIO Data Register 1	00	
	PIO Direction Control Register 1	04	
	PIO Effect Select Control Register 1	08	
	PIO New Feature Register 1	0C	

Table 1.10 Internal Address Mapping of Programmable I/O Registers

Timer Controller Registers

These registers provide user programmability between the RC32334's real-time and time-slice clock functions. Five dedicated timer functions are also programmed through the separate count, compare and control registers. Register definitions and functional overview information is provided in Chapter 16, Timer Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0700	Timer Control Register 0 (32 bits)	00	Base + Offset
	Timer Count Register 0	04	
	Timer Compare Register 0	08	
1800_0710	Timer Control Register 1 (32-bits)	00	Base + Offset
	Timer Count Register 1	04	
	Timer Compare Register 1	08	
1800_0720	Timer Control Register 2 (32-bits)	00	Base + Offset
	Timer Count Register 2	04	
	Timer Compare Register 2	08	
1800_0730	Timer Control Register 3 for WatchDog (32-bits)	00	Base + Offset
	Timer Count Register 3 for WatchDog	04	
	Timer Compare Register 3 for WatchDog	08	
1800_0740	Timer Control Register 4 for CPU BusTimeOut (BusError) (16-bits)	00	Base + Offset
	Timer Count Register 4 for CPU BusTimeOut	04	
	Timer Compare Register 4 for CPU BusTimeOut	08	

Table 1.11 Internal Address Mapping of Timer Controller Registers (Part 1 of 2)

Notes

Base Address	Register Function	Offset Address	Effective Address
1800_0750	Timer Control Register 5 for IP BusTimeout (BusError) (16-bits)	00	Base + Offset
	Timer Count Register 5 for IP BusTimeout	04	
	Timer Compare Register 5 for IP BusTimeout	08	
1800_0760	Timer Control Register 6 for DramRefresh (16-bits)	00	Base + Offset
	Timer Count Register 6 for DramRefresh	04	
	Timer Compare Register 6 for DramRefresh	08	
1800_0770	Timer Control Register 7 for WarmReset (8-bits)	00	Base + Offset
	Timer Count Register 7 for WarmReset	04	
	Timer Compare Register 7 for WarmReset	08	

Table 1.11 Internal Address Mapping of Timer Controller Registers (Part 2 of 2)

UART Control Registers

These registers enable UART functionality such as interrupt indication, data flow modes, and data receive/transmit formats. Programming the PIO controller (see Chapter 15) enables the RC32334's two identical 16550 compatible UARTs. UART registers are defined and described in more detail in Chapter 17, UART Controller.

Base Address	Register Function	Offset Address	Effective Address
UART 0			
1800_0800	Receiver Buffer Register / Transmitter Holding Register, DLAB = 0	00	Base + Offset
	Interrupt Enable Register, DLAB = 0	04	
	Baud Divisor Latch, LS, DLAB = 1	00	
	Baud Divisor Latch, MS, DLAB = 1	04	
	Interrupt Identity Register / Buffer Control Register	08	
	Line Control Register	0C	
	MODEM Control Register	10	
	Line Status Register	14	
	MODEM Status Register	18	
	Scratch Register	1C	

Table 1.12 Internal Address Mapping of UART 0 Registers

Notes

Base Address	Register Function	Offset Address	Effective Address
UART 1			
1800_0820	Receiver Buffer Register / Transmitter Holding Register, DLAB = 0	00	Base + Offset
	Interrupt Enable Register, DLAB = 0	04	
	Baud Divisor Latch, 8 LSB, DLAB = 1	00	
	Baud Divisor Latch, 8 MSB, DLAB = 1	04	
	Interrupt Identity Register / Buffer Control Register	08	
	Line Control Register	0C	
	MODEM Control Register	10	
	Line Status Register	14	
	MODEM Status Register	18	
	Scratch Register	1C	

Table 1.13 Internal Address Mapping of UART 1 Registers

Note: Table 1.13 does not apply to the RC32332.

Serial Peripheral Interface Registers

These registers enable SPI functionality. For more detailed information, see Chapter 18, “Serial Peripheral Interface.”

Base Address	Register	Offset Address	Effective Address
1800_0900	Serial Peripheral Clock Divisor/Prescalar Register (SPCNT)	00	Base + Offset
	Serial Peripheral Control Register (SPCNTL)	04	
	Serial Peripheral Status Register (SPSR)	08	
	Serial Peripheral Data I/O Register (SPDR)	0C	

Table 1.14 Internal Address Mapping of SPI Registers

DMA Control Registers

These registers determine channel usage, data transfer modes, and descriptor ownership of the four general purpose DMA channels. As programmed, each channel can move data between the source and destination ports, such as system memory, PCI, or I/O devices. Chapter 13 provides detailed programming information for the DMA registers.

Notes

Base Address	Register Function	Offset Address	Effective Address
DMA Channel 0			
1800_1400	Ch0 Configuration Register	00	Base + Offset
	Ch0 Base Descriptor Address Register	04	
	Ch0 Current Address Register	08	
	Ch0 Status/BlockSize Register	10	
	Ch0 Source Address Register	14	
	Ch0 Destination Address Register	18	
	Ch0 Next Descriptor Address Register	1C	

Table 1.15 Internal Address Mapping of DMA Channel 0 Registers

Base Address	Register Function	Offset Address	Effective Address
DMA Channel 1			
1800_1440	Ch1 ConfigurationRegister	00	Base + Offset
	Ch1 Base Descriptor Address Register	04	
	Ch1 Current Address Register	08	
	Ch1 Status/BlockSize Register	10	
	Ch1 Source Address Register	14	
	Ch1 Destination Address Register	18	
	Ch1 Next Descriptor Address Register	1C	

Table 1.16 Internal Address Mapping of DMA Channel 1 Registers

Base Address	Register Function	Offset Address	Effective Address
DMA Channel 2			
1800_1900	Ch2 Configuration Register	00	Base + Offset
	Ch2 Base Descriptor Address Register	04	
	Ch2 Current Address Register	08	
	Ch2 Status/BlockSize Register	10	
	Ch2 Source Address Register	14	
	Ch2 Destination Address Register	18	
	Ch2 Next Descriptor Address Register	1C	

Table 1.17 Internal Address Mapping of DMA Channel 2 Registers

Notes

Base Address	Register Function	Offset Address	Effective Address
DMA Channel 3			
1800_1940	Ch3 Configuration Register	00	Base + Offset
	Ch3 Base Descriptor Address Register	04	
	Ch3 Current Address Register	08	
	Ch3 Status/BlockSize Register	10	
	Ch3 Source Address Register	14	
	Ch3 Destination Address Register	18	
	Ch3 Next Descriptor Address Register	1C	

Table 1.18 Internal Address Mapping of DMA Channel 3 Registers

PCI Interface Control Registers

These registers configure system functions or modes and provide access to local memory via the PCI bus. More detailed register definitions and descriptions are provided in Chapter 12, PCI Interface Controller.

Base Address	Register Function	Offset Address	Effective Address
1800_0500	PCI Controller Interrupt Pending Register 11	B0	Base + Offset
1800_0500	PCI Controller Interrupt Mask Register 11	B4	
1800_0500	PCI Controller Interrupt Clear Register 11	B8	
1800_0500	CPU to PCI Mailbox Interrupt Pending Register 12	C0	
1800_0500	CPU to PCI Mailbox Interrupt Mask Register 12	C4	
1800_0500	CPU to PCI Mailbox Interrupt Clear Register 12	C8	
1800_0500	PCI to CPU Mailbox Interrupt Pending Register 13	D0	
1800_0500	PCI to CPU Mailbox Interrupt Mask Register 13	D4	
1800_0500	PCI to CPU Mailbox Interrupt Clear Register 13	D8	
1800_2000	New Feature Register	0A0	
1800_2000	PCI Target Control Register	0A4	
1800_2000	PCI Memory and I/O Space 1 Base Register	0B0	
1800_2000	PCI Memory and I/O Space 2 Base Register	0B8	
1800_2000	PCI Memory and I/O Space 3 Base Register	0C0	
1800_2000	PCI Memory and I/O Space 4 Base Register	0C8	
1800_2000	PCI Arbitration Register	0E0	
1800_2000	PCI CPU Space1 Base Register	0E8	
1800_2000	PCI CPU Space 2 Base Register	0F4	
1800_2000	PCI CPU Space 3 Base Register	100	

Table 1.19 Internal Address Mapping of PCI Interface Control Registers (Part 1 of 2)

Notes

Base Address	Register Function	Offset Address	Effective Address
1800_2000	PCI CPU Space 4 Base Register	10C	Base + Offset
1800_2000	PCI Configuration Address Register	CF8	
1800_2000	PCI Configuration Data Register	CFC	

Table 1.19 Internal Address Mapping of PCI Interface Control Registers (Part 2 of 2)

RC32300 CPU Core

Notes

Introduction

Targeted to a variety of software intensive embedded applications, the RC32334 is a member of the Integrated Device Technology, Inc. (IDT) RISController series of Embedded Microprocessors. It is based on the RC32300 CPU core. The RISCore 32300 CPU core continues IDT's tradition of high-performance through high-speed pipelines, high-bandwidth caches and bus interface, MIPS application specific architectural extensions.

The RC32334 supports a wide variety of embedded processor-based applications, such as communications equipment (low-end routers, gateways, switches, cellular base stations) and digital consumer systems (internet appliances).

Performance Overview

The RC32334 brings RISCore 4000 family performance levels to lower cost systems. High performance is preserved by retaining large on-chip two-way set-associative caches, a streamlined high-speed pipeline, high-bandwidth and facilities such as early restart for data cache misses.

An array of development tools as well as integrated in-circuit emulation support facilitates rapid development of RC32334-based systems, allowing a wide variety of customers to take advantage of the processor's high-performance capabilities while maintaining short time-to-market goals. Also, being upwardly software compatible with the RISCore 3000 family, the RC32334 will serve in many of the same applications. The RC32334 also supports applications that require integer digital signal processing (DSP) functions.

RC32300 CPU Core Features

- *High-performance embedded 32-bit RISCore 32300 microprocessor*
  - *Based on Enhanced MIPS-II RISC architecture*
  - *Scalar 5-stage pipeline minimizes branch and load delays*
- *Enhanced MIPS-II instruction set architecture*
  - *MIPS-IV compatible conditional move instructions*
  - *MIPS-IV superset PREF (prefetch) instruction*
  - *Fast multiplier with atomic multiply-add, multiply-sub*
  - *Count leading zero/one instructions*
- *Large, efficient on-chip caches*
  - *Separate 8kB Instruction cache and 2kB Data cache*
  - *2-way set associative*
  - *Write-back and write-through support on a per page basis*
  - *Optional cache locking, with per line resolution, to facilitate deterministic response*
  - *Simultaneous instruction and data fetch in each clock cycle*
- *Flexible RC4000 compatible MMU with 32-page TLB*
  - *Variable page size*
  - *Enhanced write algorithm support*
  - *Variable number of locked entries*
  - *No performance penalty for address translation*
- *Improved real-time support*
  - *Fast interrupt decode*
- *Low-power operation*
  - *Active power management: powers down inactive units*
- *Enhanced JTAG interface for system debug using external, low-cost in-circuit emulator (ICE) equipment*

Notes

- *On-chip debug port (compatible with EJTAG standard)*
- *MIPS architecture ensures applications software compatibility throughout the RISController series of embedded processors, and availability of a broad range of complementary hardware and software products from third parties.*

RC32300 CPU Overview

The RC32300 CPU core has a level of integration designed for high-performance and high bandwidth computing. Key elements of the CPU core are illustrated in the block diagram provided in Figure 2.1. An overview on these features follows, with more detailed explanations provided in subsequent chapters.

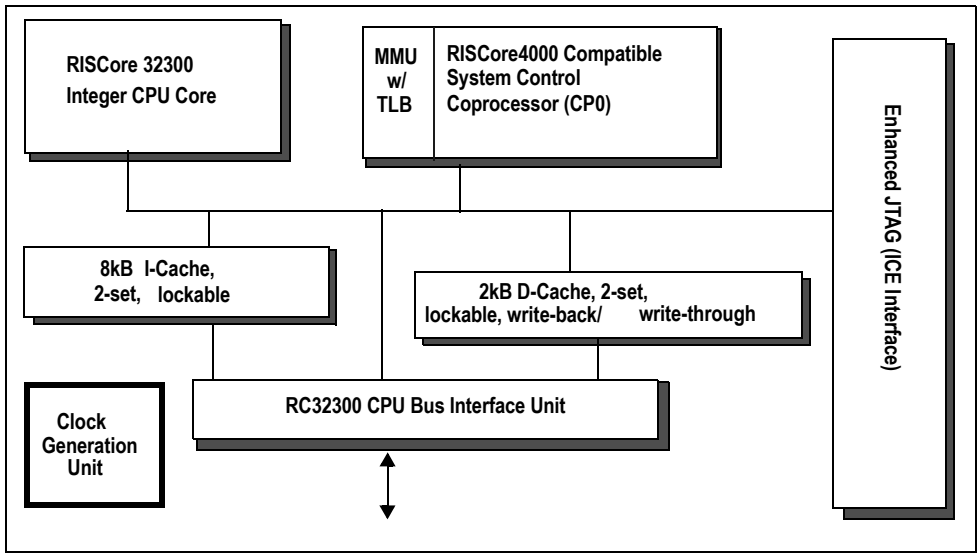


Figure 2.1 RC32300 CPU Core Block

CPU Registers

The RC32300 CPU core includes thirty-two general-purpose 32-bit registers. These registers are used for scalar integer operations and address calculation.

The register file consists of two read ports and two write ports, and it is fully bypassed to minimize operation latency in the pipeline.

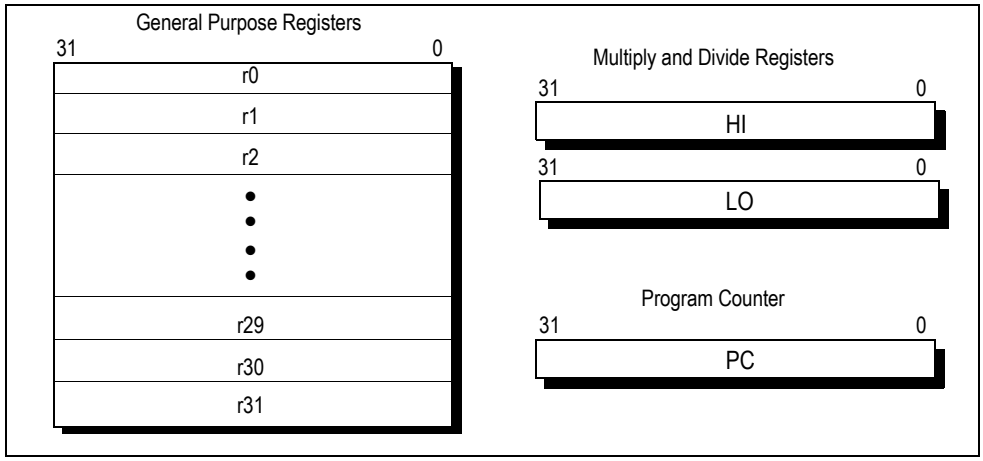


Figure 2.2 RC32300 Registers

Two of the CPU general purpose registers have the following assigned functions:

- *r0 is hardwired to a value of zero and can be used as the target register for any instruction whose result is to be discarded. r0 can also be used as a source when a zero value is required.*

Notes

- *r31 is used as an implicit return destination address register by the JAL and BAL series of instructions.*

Also, two multiply/divide registers (HI/LO) store the product of integer multiply operations or the quotient (in LO) and remainder (in HI) of integer divide operations. The RISCore 32300 CPU core does not have a Program Status Word (PSW) register, so the function traditionally covered by PSW is handled by the Cause and Status registers in the System Control Coprocessor (CPO). CPO also has a number of special purpose registers that are used in conjunction with the memory management system and during exception processing.

The RISCore 32300 implements the Enhanced MIPS-II instruction set architecture (ISA) whose features include:

- *PREF operation, with various hint subfields*
- *Conditional move instructions*
- *MAD, MUL and MSUB instructions incorporated in the integer multiply units, used to perform multiply accumulate and multiply subtract operations*
- *Count Leading Ones (CLO) and Count Leading Zeros (CLZ) instructions.*

These features come together to make the controller well suited to applications requiring the use of some DSP algorithms.

Configuration

During hardware reset, the RC32300's byte ordering is configurable into either a *big-endian* or *little-endian* convention. When configured as a big-endian system, byte 0 is always the most significant (leftmost) byte in a word (see Figure 2.3). However, when configured as a little-endian system, byte 0 is always the least significant (rightmost) byte in a word (see Figure 2.4).

Higher Address ↑ Lower Address

Big-Endian Byte Ordering

31 24	23 16	15 8	7 0	Word Address
8	9	A	B	8
4	5	8	7	4
0	1	2	3	0

* Most significant byte (MSB) is at lowest address
** Word is addressed by byte address of MSB

Figure 2.3 Big-Endian Byte Ordering Convention

Higher Address ↑ Lower Address

Little-Endian Byte Ordering

31 24	23 16	15 8	7 0	Word Address
B	A	9	8	8
7	6	5	4	4
3	2	1	0	0

* Least significant byte (LSB) is at lowest address
** Word is addressed by byte address of LSB

Figure 2.4 Little-Endian Byte Ordering Convention

Notes

CP0 Considerations

CP0 is responsible for address translation as well as cache attribute and exception management, and in the MIPS architecture, CP0 functions are allowed to vary by implementation. The RC32334 implements an RISCore 4000 family compatible CP0. Specific details on the CP0 registers implemented by the RC32300 are provided in Chapter 6, CPU Exception Processing.

Memory Management Unit (MMU)

The RC32334's MMU is modeled after the MMU found in the RISCore 4000 family and includes the Translation Lookaside Buffer (TLB). This MMU offers the following advantages, relative to the traditional RISCore 3000 family MMU:

- *Variable page size*
- *Enhanced Write Algorithm support*
- *Mapping of a larger portion of the virtual address space*
- *Variable number of locked entries*

On-chip Instruction and Data Caches

The RC32300 CPU core incorporates separate instruction (I-cache) and data caches (D-cache) that can be accessed in a single processor cycle. Each cache has its own 32-bit data path and can be accessed in the same pipeline clock cycle. The RC32300 CPU core supports a cache-locking feature, which is implemented on a "per-line basis," enabling the system designer to maximize the system's cache efficiency.

Power Reduction Mode

The RISCore 32300 is a static design and supports a WAIT instruction that is designed to signal the rest of the chip that execution and clocking should be halted, reducing system power consumption during idle periods.

Standby Mode Operation

Executing the WAIT instruction enables interrupts and enters Standby Mode. When the WAIT instruction finishes the W pipe-stage, if the bus is currently idle, the internal clocks will shut down, thus freezing the pipeline. The PLL, internal timer, and some of the input pins (cpu_nmi_n, cpu_int_n[5:4], [2:0], cpu_coldreset_n) will continue to run. If the internal IP bus is not idle when the WAIT instruction finishes the W pipe-stage, the WAIT is treated as a NOP. Once the CPU enters Standby Mode, any interrupt, including the internally generated timer interrupt, will cause the CPU to exit Standby Mode.

CPU Instruction Set Overview

Notes

Introduction

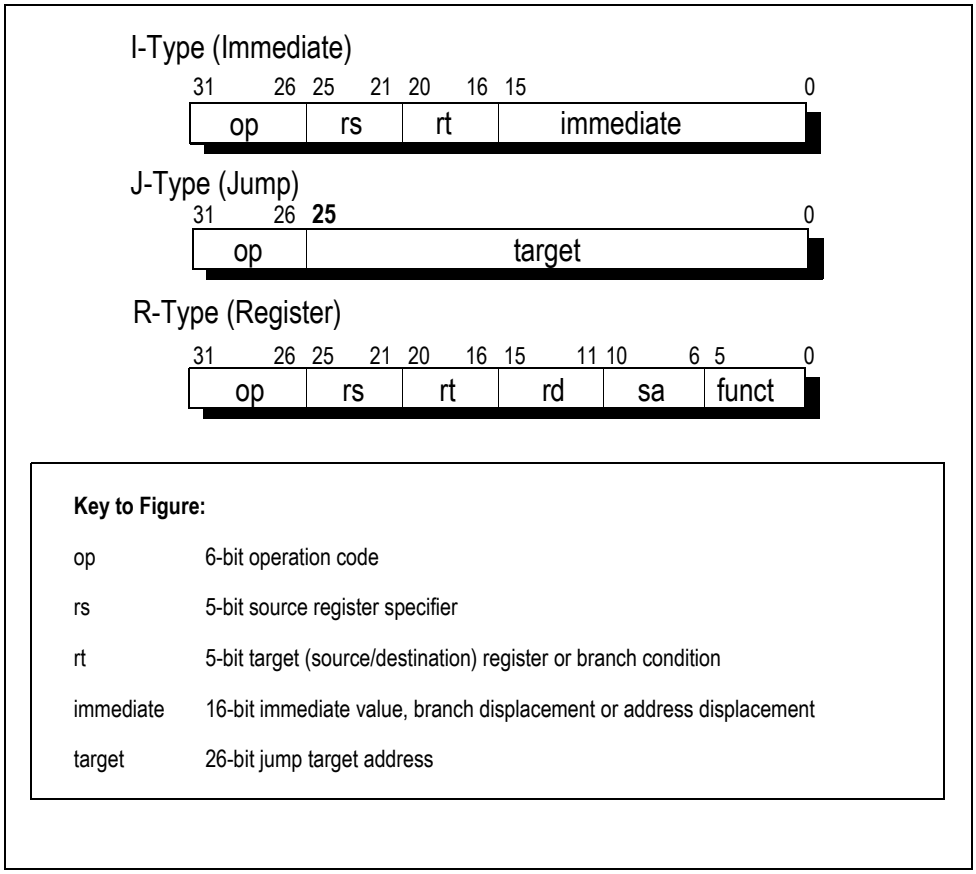
This chapter provides a general overview on the three CPU instruction set formats of the MIPS architecture: Immediate, Jump, and Register. For descriptions of the new instruction sets implemented in this device, refer to Appendix A, RC32300 CPU Core Enhancements to MIPS II ISA, in this manual. For more details on a specific CPU core instruction, refer to the *IDT MIPS Microprocessor Family Software Developer's Guide, Version 1.0*, December 1998.

CPU Instruction Formats

Each CPU instruction consists of a single 32-bit opcode, aligned on a word (4-byte) boundary. As shown in Figure 3.1, there are three CPU instruction formats:

- *Immediate (I-type)*
- *Jump (J-type)*
- *Register (R-type)*

Limiting instruction format types to three simplifies instruction decoding (thus higher frequency operations) and allows the compiler to synthesize more complicated (and less frequently used) operations and addressing modes.



Key to Figure:

op	6-bit operation code
rs	5-bit source register specifier
rt	5-bit target (source/destination) register or branch condition
immediate	16-bit immediate value, branch displacement or address displacement
target	26-bit jump target address

Figure 3.1 CPU Instruction Formats

Notes

For all MIPS processors, system control is implemented as Coprocessor 0 (CP0), the System Control Coprocessor. In the MIPS architecture, coprocessor instructions are implementation dependent. For detailed descriptions of individual Coprocessor 0 instructions, refer to the *IDT MIPS Microprocessor Family Software Developer's Guide*.

Load and Store Instructions (I-type)

Load and store are immediate (I-type) instructions that move data between memory and the general registers. The only addressing mode that load and store instructions directly support is *base register plus 16-bit signed immediate offset*.

Scheduling a Load Delay Slot

A load instruction that does not allow its result to be used by the instruction immediately following is called a *delayed load instruction*. The instruction slot immediately following this delayed load instruction is referred to as the *load delay slot*.

In the RC32334 processor, the instruction immediately following a load instruction can request the contents of the loaded register; however, in such cases, hardware interlocks may insert additional real cycles. Consequently, scheduling load delay slots can be desirable, both for performance and RC3000 processor family (e.g., R3051) compatibility. However, the scheduling of load delay slots is not absolutely required.

Defining Access Types

Access type indicates the size of an RC32334 processor data item to be loaded or stored, set by the load or store instruction opcode. Access types are defined in the *IDT MIPS Microprocessor Family Software Developer's Guide*.

Regardless of access type or byte ordering (endianness), the address given specifies the low-order byte in the addressed field. For a big-endian configuration, the low-order byte is the most-significant byte; for a little-endian configuration, the low-order byte is the least-significant byte.

The access type, together with the three low-order bits of the address, define the bytes accessed within the addressed doubleword, which is shown in Table 3.1. Only the combinations shown in this table are permitted. Other combinations will cause address error exceptions.

Access Type Mnemonic (*Value*)	Low Order Address Bits			Bytes Accessed							
				Big Endian (31.....0) Byte				Little Endian (31.....0) Byte			
	2	1	0								
Word (*3*)	0	0	0	0	1	2	3	3	2	1	0
Triplebyte (*2*)	0	0	0	0	1	2			2	1	0
	0	0	1		1	2	3	3	2	1	
Halfword (*1*)	0	0	0	0	1					1	0
	0	1	0			2	3	3	2		
Byte (*0*)	0	0	0	0							0
	0	0	1		1					1	
	0	1	0			2			2		
	0	1	1				3	3			

Table 3.1 Permitted Address Combinations

Notes

Computational Instructions (R-type and I-type)

Computational instructions can be in either the register (R-type) or immediate (I-type) formats. In the R-type format, both operands are registers; in the I-type format, one operand is a 16-bit immediate.

Computational instructions perform arithmetic, logical, shift, multiply, and divide operations on register values and fit in the following four categories:

- *ALU Immediate instructions*
- *Three-Operand Register-Type instructions*
- *Shift instructions*
- *Multiply and divide instructions*

Operations with 32-bit Operands

Operands to 32-bit operand opcodes must be in sign-extended form. 32-bit operand opcodes include all non-doubleword operations, such as: ADD, ADDU, SUB, SUBU, ADDI, SLL, SRL, SRA, SLLV, etc. The result of operations that use incorrect sign-extended 32-bit values is unpredictable.

Cycle Timing for Multiply and Divide Instructions

If necessary, RC32334 hardware *interlocks* to allow complete execution of the multiply and divide instructions. For example, MFHI and MFLO instructions are interlocked so that any attempt to read or execute them prior to the completion of previously issued multiply or divide instructions will be delayed.

Table 3.2 lists the number of processor cycles (PCycles) required to resolve an interlock or stall between various multiply or divide instructions and a subsequent MFHI or MFLO instruction. Specific details on the MFHI or MFLO instructions are provided in the *IDT MIPS Microprocessor Family Software Developer's Guide*.

Opcode	Operand Size	Latency[1]	Repeat[2]	Stall[3]
MULT/U, MAD/U, MSUB/U	16-bit	3	2	0
	32-bit	4	3	0
MUL	16-bit	3	2	1
	32-bit	4	3	2
DIV, DIVU	any	36	36	0
CLZ	32-bit	1	1	0
CLO	32-bit	1	1	0

Table 3.2 Performance Levels of MUL/DIV and New Instructions

1. Latency refers to the number of cycles before a result is available.
2. Repeat refers to the number of cycles before an operation can be re- issued.
3. Stall refers to the number of cycles that the CPU delays the pipeline.

Jump & Branch Instructions (J-type and R-type)

Jump and Branch instructions change a program's control flow. All jump and branch instructions occur with a delay of one instruction: The instruction immediately following the jump or branch (known as the instruction in the *delay slot*) always executes while the target instruction is being fetched from storage.

Overview of Jump Instructions

Subroutine calls in high-level languages are usually implemented with Jump or Jump and Link instructions, both of which are J-type instructions. In the J-type format, the 26-bit target address shifts left 2 bits and combines with the high-order 4 bits of the current program counter to form an absolute address.

Notes

Returns, dispatches, and large cross-page jumps are usually implemented with the Jump Register or Jump and Link Register instructions (both of which are R-type instructions that take the 32-bit or 64-bit byte address contained in one of the general purpose registers).

Overview of Branch Instructions

A branch instruction is a jump to a specified memory location and has an architectural delay of one instruction. All branch instruction target addresses are computed by adding the address of the instruction in the delay slot to the 16-bit *offset* (shifts left 2 bits and is sign-extended to 32 bits).

When a branch is taken, the instruction immediately following the branch instruction, in the branch delay slot, is executed before the branch to the target instruction takes place. There are two versions of Conditional branches, and each one treats the instruction in the delay slot differently. The “branch” instructions will execute the instruction in the delay slot, but the “branch likely” instructions do not. If a conditional branch likely is not taken, the instruction in the delay slot is nullified. For regular conditional branches, the delay slot is always executed.

Special Instructions (R-type)

Special instructions allow the software to initiate traps. Trap instructions cause exceptions conditionally based upon the result of a comparison. These special instructions are always R-type. For more information about special instructions, refer to the individual instruction as described in the *IDT MIPS Microprocessor Family Software Developer’s Guide*.

Exception Instructions

Exception instructions are extensions to the MIPS ISA and cause an exception that will transfer control to a software exception handler in the kernel. System call and breakpoint instructions cause exceptions unconditionally. For more information about specific exception instructions, refer to the individual instruction as described in the *IDT MIPS Microprocessor Family Software Developer’s Guide*.

Coprocessor Instructions (I-type)

Coprocessor instructions perform operations in their respective coprocessors. Coprocessor loads and stores are I-type, and coprocessor computational instructions have coprocessor-dependent formats.

CP0 instructions perform operations specifically on the System Control Coprocessor registers to manipulate the memory management and exception handling facilities of the processor.

Summary of CPU Supported Instruction Sets

The tables that follow list instructions supported by the RC32300 CPU core. Load and Store Instructions are listed in Table 3.3, Arithmetic Instructions (ALU Immediate) in Table 3.4, Arithmetic Instructions (3-Operand, R-Type) in Table 3.5, Multiply, Divide and DSP Instructions are in Table 3.6, Jump and Branch Instructions are in Table 3.7, Shift Instructions are in Table 3.8, Coprocessor Instructions are in Table 3.9, Special Instructions are listed in Table 3.10, Exception and CP0 Instructions are listed in Table 3.11 and Table 3.12.

Opcode	Description	MIPS ISA Level
LB	Load Byte	I
LBU	Load Byte Unsigned	I
LH	Load Halfword	I
LHU	Load Halfword Unsigned	I
LW	Load Word	I

Table 3.3 Load and Store Instructions (Part 1 of 2)

Notes

Opcode	Description	MIPS ISA Level
LWL	Load Word Left	I
LWR	Load Word Right	I
SB	Store Byte	I
SH	Store Halfword	I
SW	Store Word	I
SWL	Store Word Left	I
SWR	Store Word Right	I
LL	Load Linked	II
SC	Store Conditional	II
SYNC	Sync	II
PREF	Prefetch	IV

Table 3.3 Load and Store Instructions (Part 2 of 2)

Opcode	Description	MIPS ISA Level
ADDI	Add Immediate	I
ADDI	Add Immediate Unsigned	I
SLTI	Set on Less Than Immediate	I
SLTIU	Set on Less Than Immediate Unsigned	I
ANDI	AND Immediate	I
ORI	OR Immediate	I
XORI	Exclusive OR Immediate	I
LUI	Load Upper Immediate	I

Table 3.4 Arithmetic Instructions (ALU Immediate)

Opcode	Description	MIPS ISA Level
ADD	Add	I
ADDU	Add Unsigned	I
SUB	Subtract	I
SUBU	Subtract Unsigned	I
SLT	Set on Less Than	I
SLTU	Set on Less Than Unsigned	I
AND	AND	I
OR	OR	I
XOR	Exclusive OR	I
NOR	NOR	I
MOVN	Move Conditional on Not Zero	IV
MOVZ	Move Conditional on Zero	IV

Table 3.5 Arithmetic Instructions (3-Operand, R-Type)

Notes

Opcode	Description	MIPS ISA Level
MULT	Multiply	I
MULTU	Multiply Unsigned	I
DIV	Divide	I
DIVU	Divide Unsigned	I
MFHI	Move From HI	I
MTHI	Move To HI	I
MFLO	Move From LO	I
MTLO	Move To LO	I
MUL	Multiply with destination register writeback	RC32364, RC4650, RISCore 32300, RC64574/575
MAD	Multiply Add	RC32364, RC4650, RISCore 32300, RC64574/575
MADU	Multiply Add Unsigned	RC32364, RC4650, RISCore 32300, RC64574/575
MSUB	Multiply Subtract	RISCore 32300, RC32364, RC64574/575
MSUBU	Multiply Subtract Unsigned	RISCore 32300, RC32364, RC64574/575
CLZ	Count Leading Zeros	RISCore 32300, RC32364, RC64574/575
CLO	Count Leading Ones	RISCore 32300, RC32364, RC64574/575

Table 3.6 Multiply, Divide and DSP Instructions

Opcode	Description	MIPS ISA Level
J	Jump	I
JAL	Jump And Link	I
JR	Jump Register	I
JALR	Jump And Link Register	I
BEQ	Branch on Equal	I
BNE	Branch on Not Equal	I
BLEZ	Branch on Less Than or Equal to Zero	I
BGTZ	Branch on Greater Than Zero	I
BLTZ	Branch on Less Than Zero	I
BGEZ	Branch on Greater Than or Equal to Zero	I
BLTZAL	Branch on Less Than Zero and Link	I
BGEZAL	Branch on Greater Than or Equal to Zero and Link	I
BCzT	Branch on Coprocessor z True	I
BCzF	Branch on Coprocessor z False	I
BEQL	Branch on Equal Likely	II
BNEL	Branch on Not Equal Likely	II
BLEZL	Branch on Less Than or Equal to Zero Likely	II
BGTZL	Branch on Greater Than Zero Likely	II
BLTZL	Branch on Less Than Zero Likely	II

Table 3.7 Jump and Branch Instructions (Part 1 of 2)

Notes

Opcode	Description	MIPS ISA Level
BGEZL	Branch on Greater Than or Equal to Zero Likely	II
BLTZALL	Branch on Less Than Zero And Link Likely	II
BGEZALL	Branch on Greater Than or Equal to Zero and Link Likely	II
BCzTL	Branch on Coprocessor z True Likely	II
BCzFL	Branch on Coprocessor z False Likely	II

Table 3.7 Jump and Branch Instructions (Part 2 of 2)

Opcode	Description	MIPS ISA Level
SLL	Shift Left Logical	I
SRL	Shift Right Logical	I
SRA	Shift Right Arithmetic	I
SLLV	Shift Left Logical Variable	I
SRLV	Shift Right Logical Variable	I
SRAV	Shift Right Arithmetic Variable	I

Table 3.8 Shift Instructions

Opcode	Description	MIPS ISA Level
LWCz	Load Word to Coprocessor z	I
SWCz	Store Word from Coprocessor z	I
MTCz	Move To Coprocessor z	I
MFCz	Move From Coprocessor z	I
CTCz	Move Control To Coprocessor z	I
CFCz	Move Control From Coprocessor z	I
COPz	Coprocessor Operation z	I

Table 3.9 Coprocessor Instructions

Opcode	Description	MIPS ISA Level
SYSCALL	System Call	I
BREAK	Break	I
DRET	Debug Exception Return	RC32364, RISCore 32300
SDBBP	Software Debug Breakpoint	RC32364, RISCore 32300

Table 3.10 Special Instructions

Notes

Opcode	Description	MIPS ISA Level
TGE	Trap if Greater Than or Equal	II
TGEU	Trap if Greater Than or Equal Unsigned	II
TLT	Trap if Less Than	II
TLTU	Trap if Less Than Unsigned	II
TEQ	Trap if Equal	II
TNE	Trap if Not Equal	II
TGEI	Trap if Greater Than or Equal Immediate	II
TGEIU	Trap if Greater Than or Equal Immediate Unsigned	II
TLTI	Trap if Less Than Immediate	II
TLTIU	Trap if Less Than Immediate Unsigned	II
TEQI	Trap if Equal Immediate	II
TNEI	Trap if Not Equal Immediate	II

Table 3.11 Exception Instructions

Opcode	Description	MIPS ISA Level
MTC0	Move To CP0	I
MFC0	Move From CP0	I
TLBR	Read Indexed TLB Entry	I
TLBWI	Write Indexed TLB Entry	I
TLBWR	Write Random TLB Entry	I
TLBP	Probe TLB for Matching Entry	I
CACHE	Cache Operation	RISCore 4000, RISCore 32300
ERET	Exception Return	RISCore 4000, RISCore 32300
WAIT	Enter Standby mode	RISCore 4000, RISCore 32300

Table 3.12 CP0 Instructions

CPU Pipeline Architecture

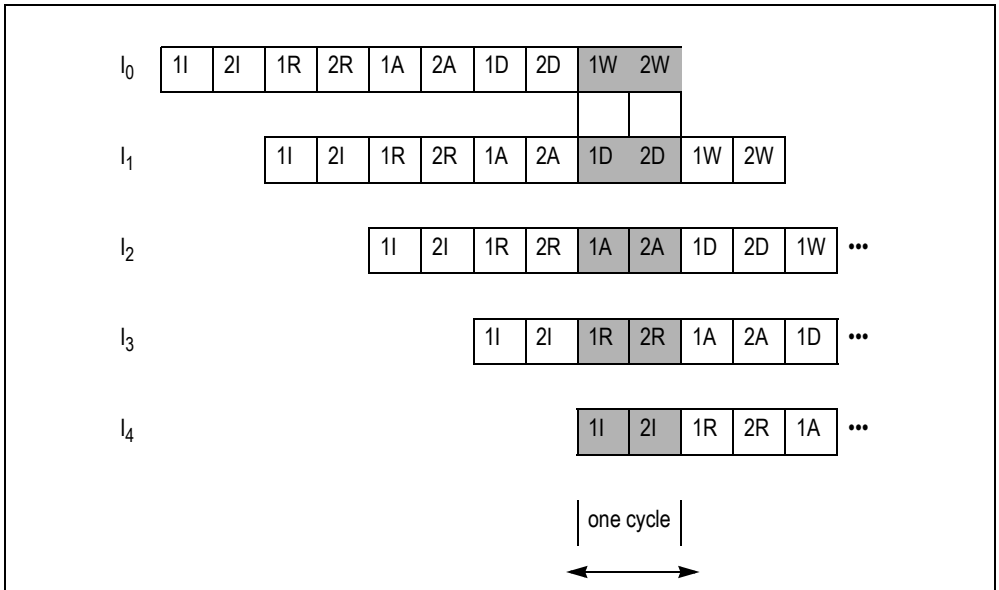
Notes

Introduction

The RISCore 32300 uses a 5-stage instruction pipeline, similar to the RISCore 3000 and RISCore 4000 families. The simplicity of this pipeline enables the processor to achieve high frequency while minimizing device complexity. The RISCore 32300 core supports limited out-of-order execution.

The RISCore 32300 pipeline also performs virtual-to-physical address translation in parallel with cache access. Additional enhancements such as prefetch operations and two new instructions allow the RC32334 to be a lower cost and lower power device than super-scalar or super-pipelined processors.

The 5-stage instruction pipeline is illustrated in Figure 4.1.



Key to Figure:

1I-1R	Instruction cache access	2R	Instruction decode
1I-2I	Instruction virtual to physical address translation	1A-2A	Integer add, logical, shift
2A-2D	Data cache access and load align	1A	Data virtual address calculation
1D-2D	Data virtual to physical address translation	2A	Store align
2R	Register file read	1A	Branch decision
2R	Bypass calculation	2W	Register file write

Figure 4.1 Instruction Pipeline Stages

CPU Pipeline Stages

This section describes each of the phases of the five pipeline stages. Each stage has 2 phases:

- *1I - Instruction Fetch, Phase one*
- *2I - Instruction Fetch, Phase two*
- *1R - Register Fetch, Phase one*

Notes

- *2R - Register Fetch, Phase two*
- *1A - Execution, Phase one*
- *2A - Execution, Phase two*
- *1D - Data Fetch, Phase one*
- *2D - Data Fetch, Phase two*
- *1W - Write Back, Phase one*
- *2W - Write Back, Phase two*

1I - Instruction Fetch, Phase One

The instruction address translation begins during the 1I phase.

2I - Instruction Fetch, Phase Two

During the 2I phase, the instruction cache fetch begins and the instruction address translation continues.

1R - Register Fetch, Phase One

During the 1R phase, the following occurs:

- *The instruction cache fetch finishes.*
- *The instruction cache tag is checked against the physical page frame number obtained from the address translation.*

2R - Register Fetch, Phase Two

During the 2R phase, the following occurs:

- *The instruction decoder decodes the instruction.*
- *Any required operands are fetched from the register file.*
- *Make a decision to either issue or slip (for an interlock condition).*
- *For a branch, the branch address is calculated.*

1A - Execution, Phase One

During the 1A phase, one of the following occurs:

- *Any result from the A or D stages are bypassed.*
- *The arithmetic logic unit (ALU) starts the integer arithmetic, logical or shift operation.*
- *The ALU calculates the data virtual address for load and store instructions.*
- *The ALU determines whether the branch condition is true.*

2A - Execution, Phase Two

During the 2A phase, one of the following occurs:

- *The integer arithmetic, logical or shift operation will complete.*
- *A data cache access will start.*
- *Store data is shifted to the specified byte position(s).*
- *The data virtual to physical address translation will start.*

1D - Data Fetch, Phase One

During the 1D phase, one of the following occurs:

- *The data cache access will continue.*
- *The data address translation completes.*

Notes

2D - Data Fetch, Phase Two

During the 2D phase, the data cache access will finish and the data is then shifted down and extended. The data cache tag is checked against the physical address for any data cache access.

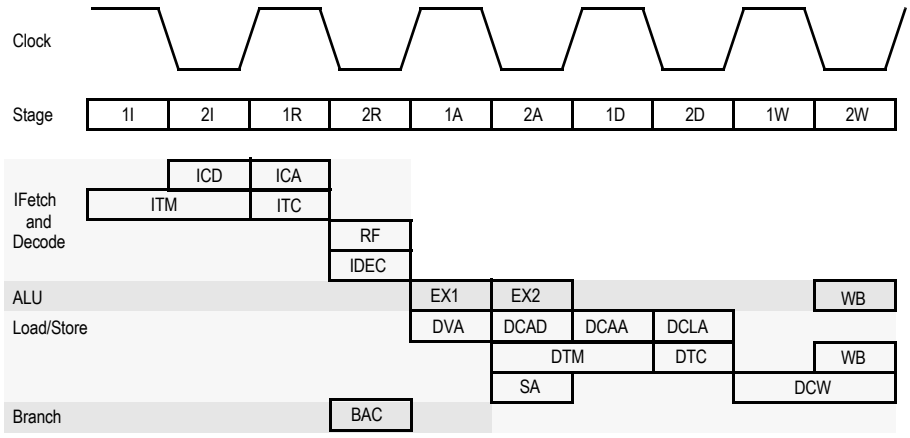
1W - Write Back, Phase One

The processor uses this phase internally to resolve all exceptions in preparation for the register file write.

2W - Write Back, Phase Two

For register-to-register and load instructions, the result is written back to the register file during the 2W stage. Branch instructions perform no operation during this stage.

Figure 4.2 shows the activities occurring during each ALU pipeline stage, for load, store, and branch instructions.



Key to Figure:

ICD	Instruction cache address decode	ICA	Instruction cache array access
ITM	Instruction translation match	RF	Register operand fetch
ITC	Instruction tag check	EX1	Operation stage 1
IDEC	Instruction decode	WB	Write back to register file
EX2	Operation stage 2	DCAD	Data cache address decode
DVA	Data virtual address calculation	DCLA	Data cache load align
DCAA	Data cache array access	DTM	Data translation match
DTC	Data tag check	SA	Store align
DCW	Data cache write	BAC	Branch address calculation

Figure 4.2 Pipeline Activities

Branch Delay

The CPU pipeline has a branch delay of one cycle and a load delay of one cycle. The one-cycle branch delay is a result of the branch decision logic operating during the 1A pipeline phase of the branch instruction. This allows the branch target address calculated in the previous phase to be used for the instruction access in the following “1I” phase.

Notes

The pipeline will begin the fetch of the branch path as well as the fall-through path in the cycle following the delay slot. After the branch decision is made, the processor will continue with the fetch of either the branch path (for a taken branch) or the fall-through path (for the non-taken branch).

Figure 4.3 illustrates the branch delay.

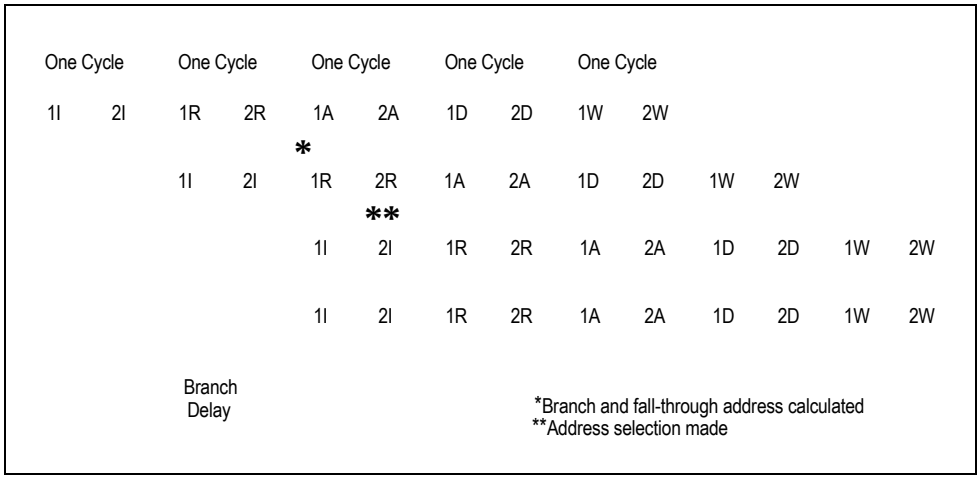


Figure 4.3 CPU Pipeline Branch Delay

Load Delay

The completion of a load at the end of the 2D pipeline phase produces an operand that is available for the 1A pipeline phase of the instruction following the load delay slot.

Figure 4.4 shows the load delay of one pipeline cycle.

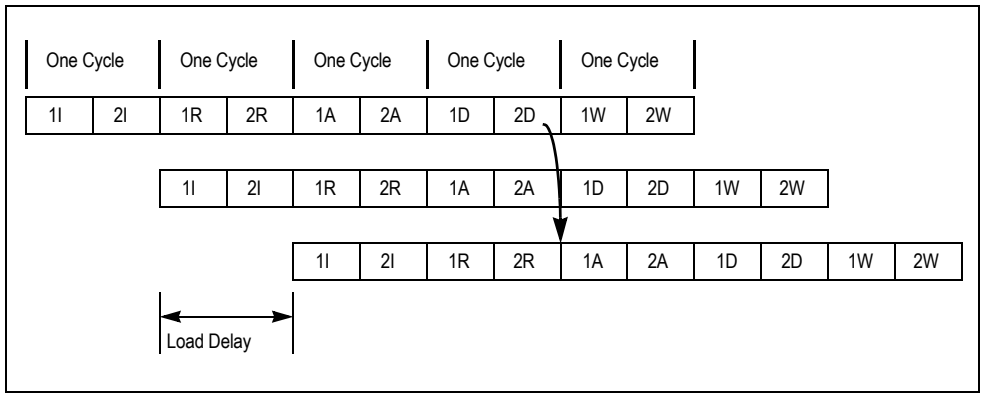


Figure 4.4 CPU Pipeline Load Delay

Interlock and Exception Handling

When cache misses or exceptions occur or when data dependencies are detected, smooth pipeline flow is interrupted. These interruptions are either handled through hardware or software methods. Software-managed interruptions are known as exceptions; hardware-handled interruptions—such as cache misses—are referred to as interlocks and occur as either stalls or slips.

Resolving a stall requires halting the pipeline; slips require the back end of the pipeline to advance while the front end of the pipeline is held static.

During all active instructions, exception and interlock conditions are checked for at each pipeline cycle. Because each exception or interlock condition corresponds to a particular pipeline stage, a condition can be traced back to the particular instruction in the exception/interlock stage. For instance, a Reserved Instruction (RI) exception is raised in the execution (A) stage.

Notes

Exception Conditions

When an exception condition occurs, the relevant instruction and all instructions that follow are cancelled. Accordingly, any stall conditions—and any later exception conditions that may have referenced this instruction—are inhibited; there is no benefit in servicing stalls for a cancelled instruction.

When an exceptional condition is detected for an instruction, the RC32334 kills it and all instructions that follow. When this instruction reaches the W stage, the exception flag causes it to write various CP0 registers with the exception state, change the current PC to the appropriate exception vector address, and clear the exception bits of earlier pipeline stages.

This implementation allows all preceding instructions to complete execution and prevents all subsequent instructions from completing. Thus, the value in the EPC is sufficient to restart execution. It also ensures that exceptions are taken in the order of execution; an instruction taking an exception may itself be killed by an instruction further down the pipeline that takes an exception in a later cycle.

Figure 4.5 shows the exception detection procedure (for example, a reserved instruction exception).

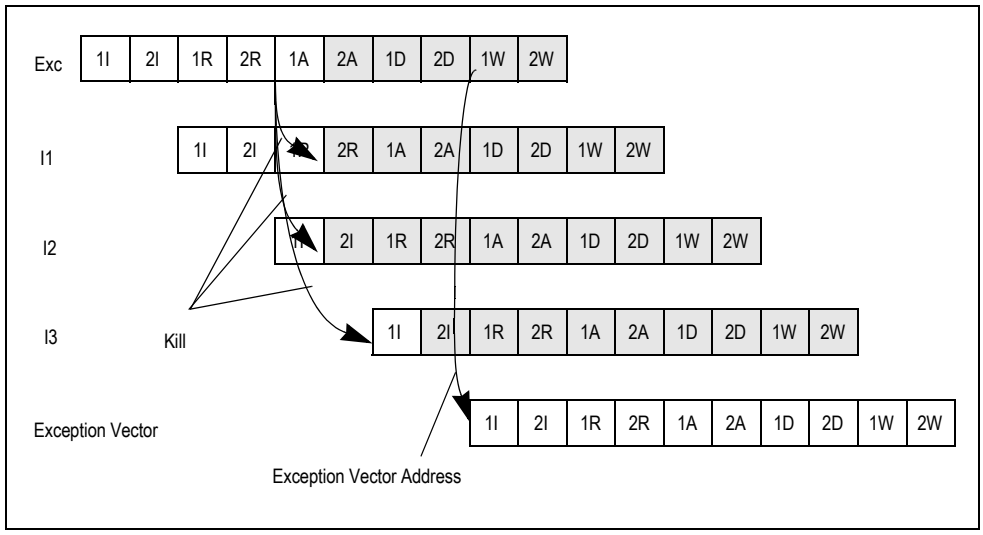


Figure 4.5 Exception Detection

Stall Conditions

Stalls are used to stop the pipeline for conditions detected after the R pipe-stage. When a stall occurs, the processor will resolve the condition and then the pipeline will continue. Figure 4.6 illustrates a data cache miss stall.

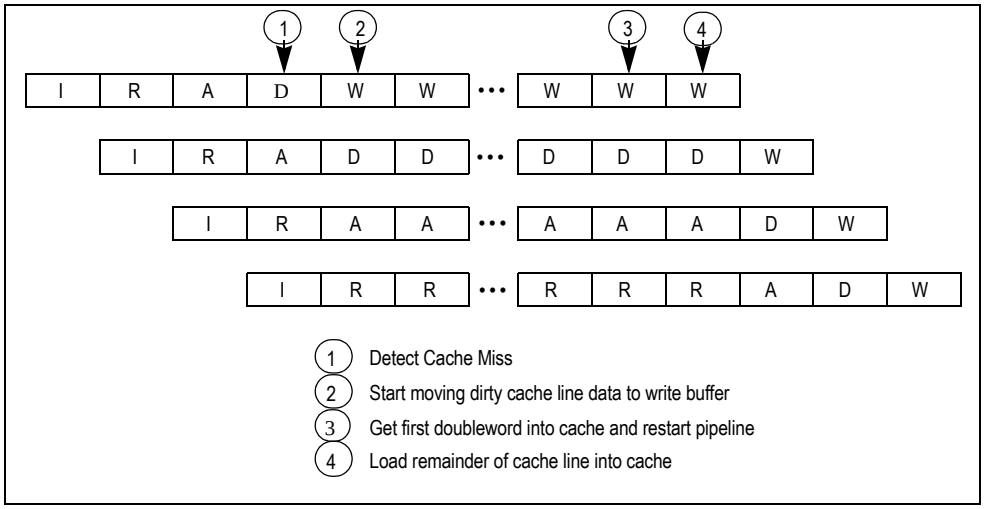


Figure 4.6 Data Cache Miss

Notes

As shown, the data cache miss is detected in the D pipe stage. If the cache line to be replaced is dirty—the W bit is set—the data is moved to the internal write buffer in the next cycle.

The first doubleword of data is returned to the cache in 3 and the pipeline will then restart. The remainder of the cache line is returned in the subsequent cycles. The data to be written back will be returned to memory some time after the entire new cache line is returned.

Slip Conditions

During the 2R and 1A pipe-stages, internal logic will determine whether it is possible to start the current instruction in this cycle. If all of the source operands are available (either from the register file or via the internal bypass logic) and all the hardware resources necessary to complete the instruction will be available at the necessary time(s), then the instruction "issues"; otherwise, the instruction will "slip".

Slipped instructions are retried on subsequent cycles until they issue. The back end of the pipeline (stages D and W) will advance normally during slips in an attempt to resolve the conflict. "NOPS" will be inserted into the bubble in the pipeline. Instructions killed by branch likely instructions, ERET or exceptions will not cause slips. Figure 4.7 shows an instruction cache miss.

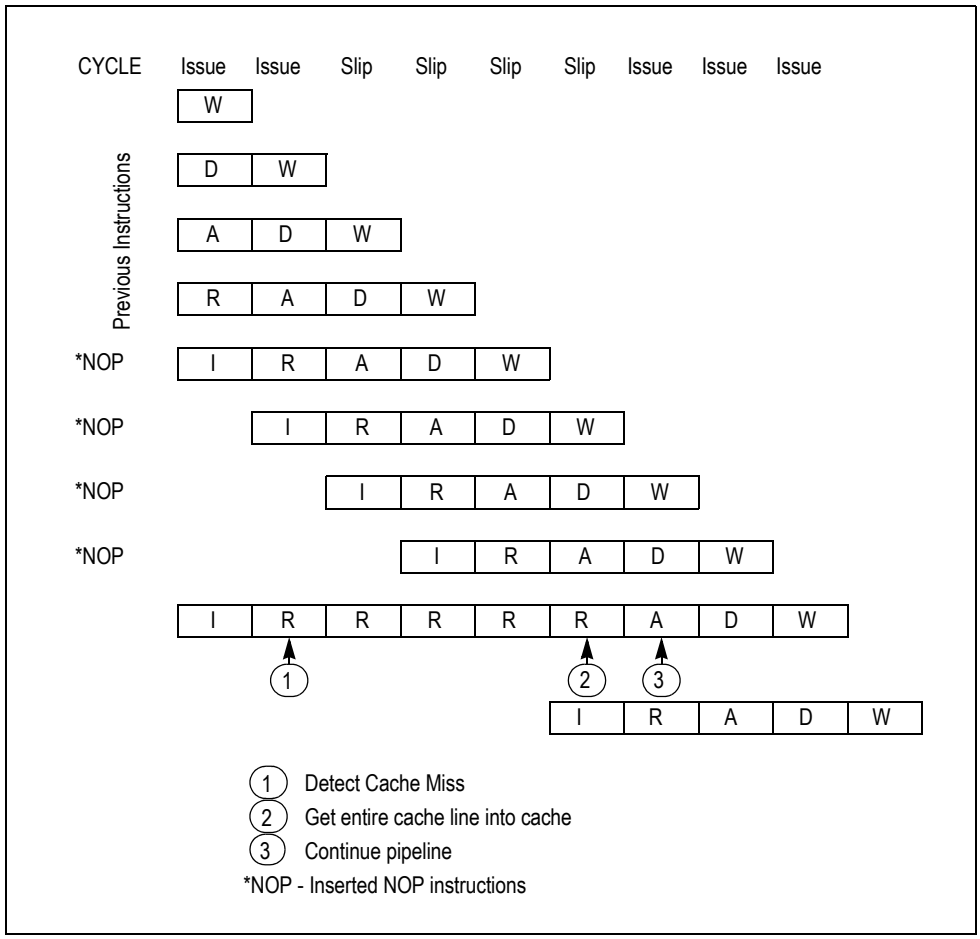


Figure 4.7 Instruction Cache Miss

As shown in Figure 4.7, instruction cache misses are detected in the R stage and the pipeline slips in its A stage. There can never be a write-back required for an instruction cache miss since dirty data can not exist in the I cache.

Writes are not allowed to the I-cache. Note that early restart is not employed for instruction cache misses. The requested cache line will be loaded into the cache in its entirety and, after that, the pipeline will restart.

Memory Management

Notes

Introduction

The Memory Management Unit (MMU) of the RC32334 is modeled after the MMU found in the R4000 families and generates typical translation lookaside buffer (TLB) exceptions such as TLB refill, TLB invalid, and TLB modified to the Integer Unit and offers the following advantages (relative to the traditional 32-bit R3000 style MMU):

- *Variable page size*
- *Enhanced Write Algorithm support*
- *Mapping of a larger portion of the virtual address space*
- *Variable number of locked entries*

Virtual-to-Physical Address Translation

Figure 5.1 illustrates the virtual-to-physical address translation of a 32-bit virtual address. The top section of the drawing shows a virtual address with a 12-bit—or 4kbyte—page size labelled *Offset*. The remaining 20 bits of the address represent the virtual page number (VPN) and index the 1M-entry page table.

The lower section of the drawing shows a virtual address with a 24-bit—or 16Mbyte—page size labelled *Offset*. The remaining 8 bits of the address represent the VPN and index a 256-entry memory-resident page table.

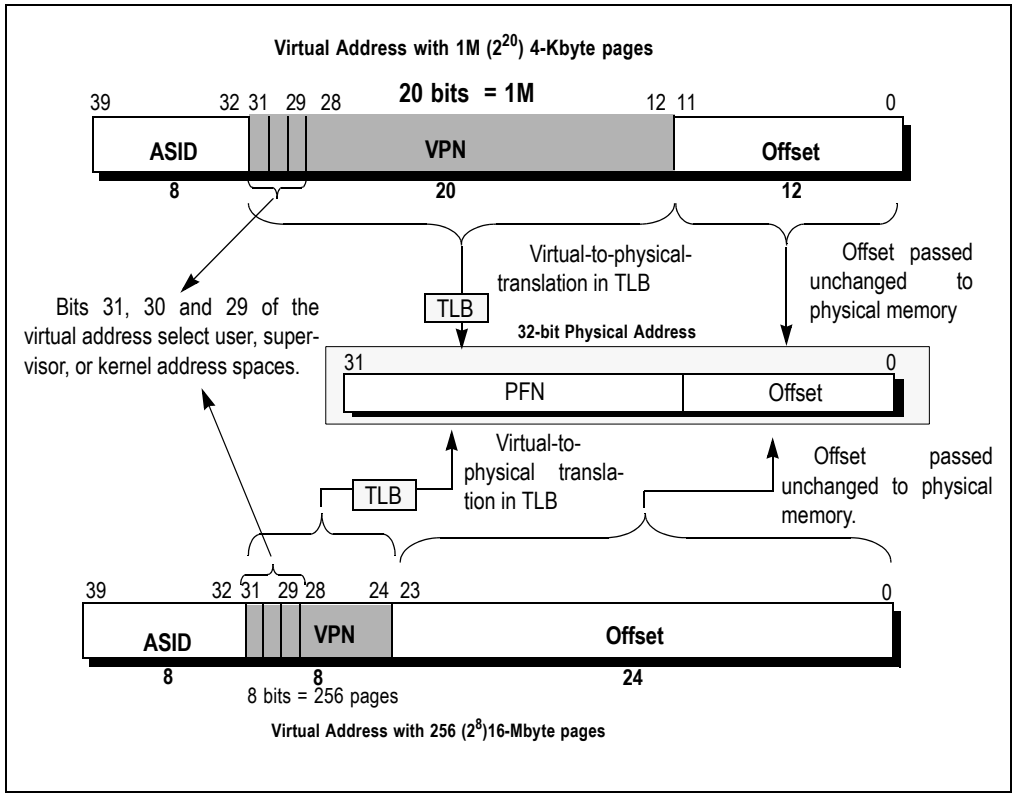


Figure 5.1 Overview of a 32-bit Virtual Address Translation

Notes

TLB Management

For fast virtual-to-physical address decoding, the RC32334 TLB is a fully associative on-chip memory device that contains 16 entries, to provide mapping to 16 odd and even page pairs of sizes varying from 4 KBytes to 16 MBytes. Each entry logically occupies a portion of a 128-bit frame work. Each field of a TLB entry has a corresponding field in the EntryHi, EntryLo0, EntryLo1, or PageMask registers.

The RC32334's TLB also contains information to control the cache coherency protocol for each page. Specifically, each page has attribute bits to determine whether the coherency algorithm is uncached, noncoherent write-back, or non-coherent write-through no write-allocate.

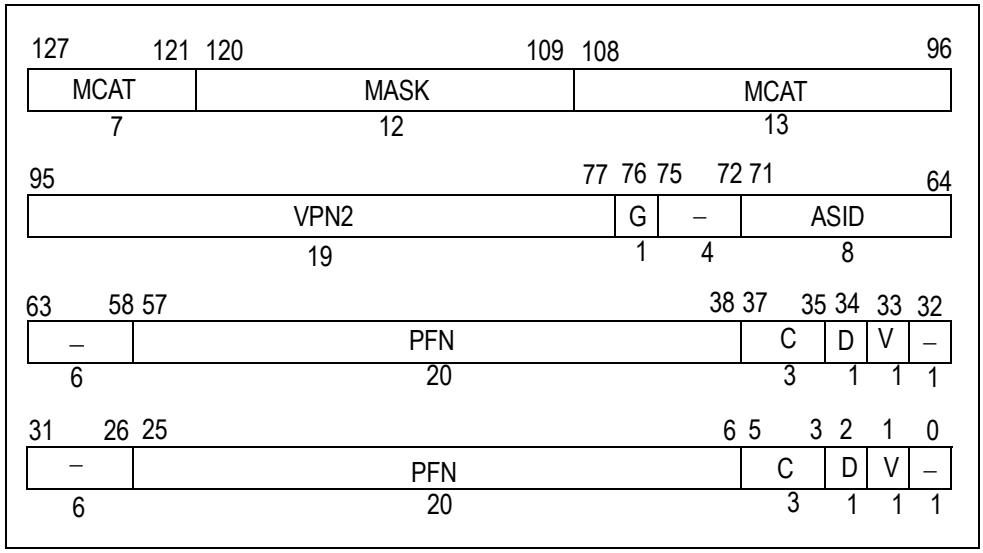


Figure 5.2 TLB Register Format

Field	Description
MASK	Page comparison mask
VPN2	Virtual Page Number divided by two (maps to two pages)
ASID	Address Space ID
G	Global. If this bit set, then ignore the ASID
PFN	Page Frame Number. Upper bit of physical address
C	Specifies the Cache Algorithm to be used, as shown below:
D	Dirty bit. This bit serves as a "write protect" bit
V	Valid bit. It set, TLB is valid. Otherwise a TLB Miss occurs
MCAT	Memory Controller Attributes. Reserved in RC32334 and must be written as '0'.

C Value	Page Coherency Attribute
0	Cacheable, noncoherent, write-through, no write allocate
1	Cacheable, noncoherent, write-through, write allocate
2	Uncached
3	Cacheable, noncoherent, write-back
4:7	Reserved

Table 5.1 TLB Register Field Descriptions

Notes

MMU Register Descriptions

The CP0 registers required to implement the RC32334 memory management unit are listed in Table 5.2. For each register, format illustrations and complete descriptions follow the table.

Number	Register	Description
0	Index	Programmable pointer into TLB array
1	Random	Pseudorandom pointer into TLB array (read only)
2	EntryLo0	Low half of TLB entry for even virtual page (VPN)
3	EntryLo1	Low half of TLB entry for odd virtual page (VPN)
4	Context	Pointer to kernel virtual page table entry (PTE)
5	PageMask	TLB Page Mask to support variable page size.
6	Wired	Number or wired TLB entries
8	BadVaddr	Bad Virtual Address
10	Entry Hi	Holds the high-order bits of a TLB entry for TLB read and write operations and is accessed by the TLB Probe, TLB Write Random, TLB Write Indexed, and TLB Read Indexed instructions.

Table 5.2 RC32334 MMU Registers

Index Register (0)

The *Index* register is a 32-bit, read/write register containing six bits to index an entry in the TLB. The high-order bit of the register shows the success or failure of a TLB Probe (TLBP) instruction. The *Index* register also specifies the TLB entry affected by TLB Read (TLBR) or TLB Write Index (TLBWI) instructions.

Note that the RC32334 contains a 16 entry TLB, while the Index register contains the capability to point to 64 TLB entries. In programming, the value written to the Index register must be in the valid range of the number of entries of the current device.

RC32334 implements additional bits in anticipation of derivative products. Figure 5.3 shows the format of the *Index* register; Table 5.3, which follows the figure, describes the contents of the *Index* register fields.

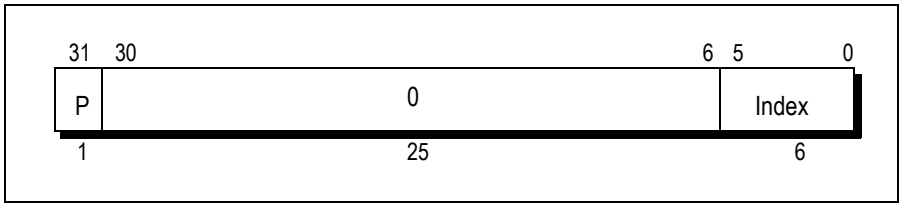


Figure 5.3 Index Register Format

Field	Description
P	Probe failure. Set to 1 when the previous TLBProbe (TLBP) instruction was unsuccessful.
Index	Index to the TLB entry affected by the TLBRead and TLBWrite instructions
0	Reserved. Must be written as zeroes, returns zeroes when read.

Table 5.3 Index Register Field Descriptions

Notes

Random Register (1)

The *Random* register is a read-only register of which 4 bits index an entry in the TLB. This register decrements as each instruction executes, and its values range between an upper and a lower bound, as follows:

- *A lower bound is set by the number of TLB entries reserved for exclusive use by the operating system (the contents of the Wired register).*
- *An upper bound is set by the total number of TLB entries. Thus the upper bound is 15 (The TLB entries are numbered from 0 to 15).*

Note: *The RC32334 implements this register differently from the 64-bit family of RISControllers. The RC4000, RC5000, and RC645xx CPUs count both valid and invalid instructions. However, the RC32334 counts only valid instructions.*

The *Random* register specifies the entry in the TLB that is affected by the TLB Write Random instruction. The register does not need to be read for this purpose (it is implicit in the instruction itself); however, the register is readable to verify proper operation of the processor.

To simplify testing, the *Random* register is set to the value of the upper bound upon system reset. This register is also set to the upper bound when the *Wired* register is written. Figure 5.4 shows the format of the *Random* register. Table 5.4 describes the contents of the *Random* register fields.

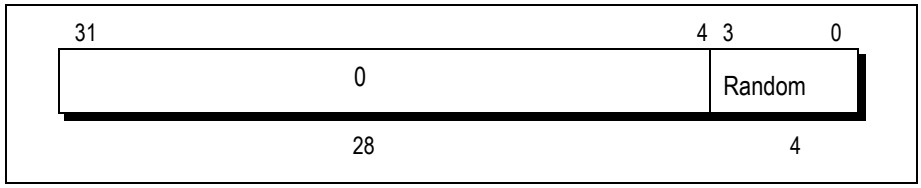


Figure 5.4 Random Register Format

Field	Description
Random	TLB random index
0	Reserved. Must be written as zeroes, and returns zeroes when read.

Table 5.4 Random Register Field Descriptions

EntryLo0 (2), and EntryLo1 (3) Registers

The *EntryLo* register consists of two registers with identical formats:

- *EntryLo0 is used for even virtual pages.*
- *EntryLo1 is used for odd virtual pages.*

The *EntryLo0* and *EntryLo1* registers are read/write registers. They hold the physical page frame number (PFN) of the TLB entry for even and odd pages, respectively, when performing TLB read and write operations.

Figure 5.5 shows the format of this register. Table 5.5 provides descriptions for the fields of this register.

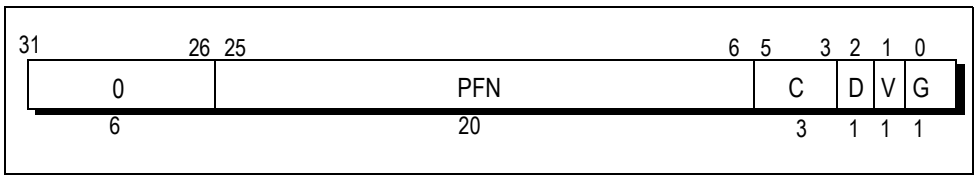


Figure 5.5 EntryLo0 and EntryLo1 Register Formats

Notes

Field	Description
PFN	Page frame number: the upper bits of the physical address.
C	Specifies the TLB page coherency attribute.
D	Dirty. If this bit is set, the page is marked as dirty and, therefore, writable. This bit is actually a write-protect bit that software can use to prevent alteration of data.
V	Valid. If this bit is set, it indicates that the TLB entry is valid; otherwise, a TLBL or TLBS miss occurs.
G	Global. If this bit is set in both Lo0 and Lo1, then the processor ignores the ASID during TLB lookup.
0	Reserved. Must be written as zeroes, returns zeroes when read.

Table 5.5 EntryLo0 and EntryLo1 Register Field Descriptions

The TLB page coherency attribute (*C*) bits specify whether references to the page should be cached. If cached, the algorithm selects between several coherency attributes.

Table 5.6 lists the coherency attributes that can be selected by the C bits.

C Value	Page Coherency Attribute
0	Cacheable, noncoherent, write-through, no write allocate
1	Cacheable, noncoherent, write-through, write allocate
2	Uncached
3	Cacheable, noncoherent, write-back
4:7	Reserved

Table 5.6 TLB Page Coherency Attributes

Context Register (4)

The *Context* register is a read/write register that contains the pointer to an entry in the page table entry (PTE) array. This array is an operating system data structure that stores virtual-to-physical address translations. When there is a TLB miss, the CPU loads the TLB with the missing translation from the PTE array.

Normally, the operating system uses the *Context* register to address the current page map that resides in the kernel-mapped segment. The *Context* register duplicates some of the information provided in the *BadVAddr* register, but the information is arranged in a form that is more useful for a software TLB exception handler.

Figure 5.6 illustrates the format of the *Context* register. Table 5.7 provides the descriptions of the *Context* register fields.

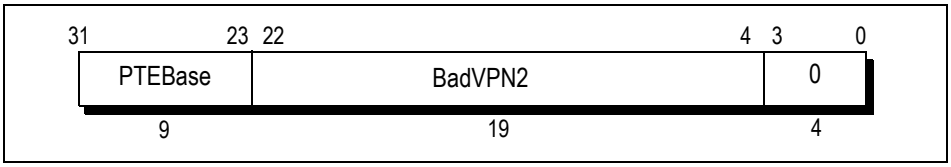


Figure 5.6 Context Register Format

Field	Description
BadVPN2	This field is written by hardware on a miss. It contains the virtual page number (VPN) pair of the most recent virtual address that did not have a valid translation.
PTEBase	This field is a read/write field for use by the operating system. It is normally written with a value that allows the operating system to use the *Context* register as a pointer into the current PTE array in memory.

Table 5.7 Context Register Field Descriptions

Notes

The 19-bit *BadVPN2* field contains bits 31:13 of the virtual address that caused the TLB miss. Bit 12 is excluded because a single TLB entry maps to an even/odd page pair. For a 4-Kbyte page size, this format can directly address the pair-table of 8-byte PTEs. For other page and PTE sizes, shifting and masking this value produces the appropriate address.

PageMask Register (5)

The *PageMask* register is a read/write register used for reading from or writing to the TLB; it holds a comparison mask that sets the variable page size for each TLB entry, as shown in the following table.

TLB read and write operations use this register as either a source or a destination. When virtual addresses are presented for translation into physical address, the corresponding bits in the TLB identify which virtual address bits, among bits 24:13, are to be used in the comparison. When the *Mask* field is not one of the values shown below, the operation of the TLB is undefined.

PageSize	Bit											
	24	23	22	21	20	19	18	17	16	15	14	13
4 Kbytes	0	0	0	0	0	0	0	0	0	0	0	0
16 Kbytes	0	0	0	0	0	0	0	0	0	0	1	1
64 Kbytes	0	0	0	0	0	0	0	0	1	1	1	1
256 Kbytes	0	0	0	0	0	0	1	1	1	1	1	1
1 Mbyte	0	0	0	0	1	1	1	1	1	1	1	1
4 Mbytes	0	0	1	1	1	1	1	1	1	1	1	1
16 Mbytes	1	1	1	1	1	1	1	1	1	1	1	1

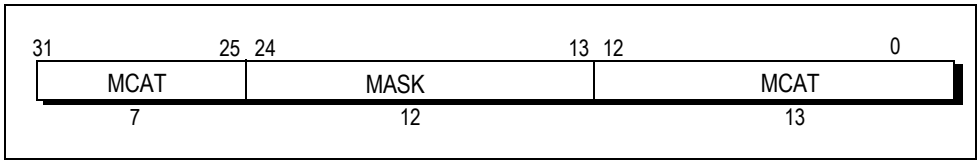


Figure 5.7 PageMask Register Format

Field	Description
Mask	Page comparison mask
MCAT	Memory controller attributes.

Table 5.8 PageMask Register Field Descriptions

Note: For the RC32334 the Memory Controller Attributes (MCAT) fields perform no user valid function. For this device, these bit fields must be written as '0'.

Wired Register (6)

The *Wired* register is a read/write register that specifies the boundary between the *wired* and *random* entries of the TLB, as shown in Figure 5.8. "Wired" entries are nonreplaceable entries, which cannot be overwritten by a TLB write random operation. "Random" entries can be overwritten. Thus, the Wired register specifies the smallest value taken by the Random register.

Note: The Index register is not affected by the Wired register. The Index register can still point to and be used to overwrite either "Random" or "Wired" TLB entries.

Notes

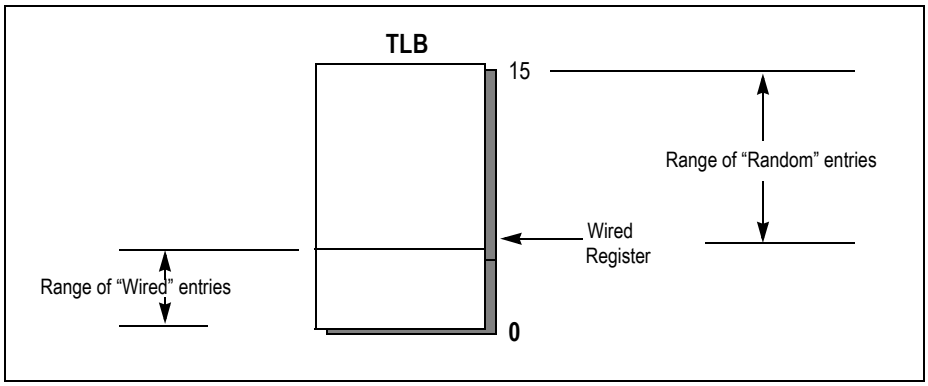


Figure 5.8 Diagram Showing Ranges of Wired and Random Entries

The *Wired* register is set to 0 upon system reset. Writing to this register also sets the *Random* register to the value of its upper bound (see *Random* register format in Figure 5.4 and Table 5.4). Figure 5.9 shows the format of the *Wired* register, and Table 5.9 lists the contents of this register's fields.

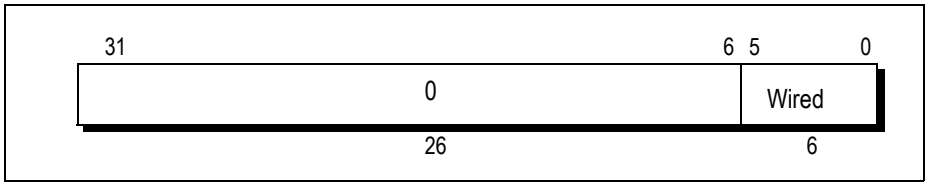


Figure 5.9 Wired Register Format

Field	Description
Wired	TLB Wired boundary (the number of wired TLB entries)
0	Reserved. Must be written as zeroes, and returns zeroes when read.

Table 5.9 Wired Register Field Descriptions

Note that the RISCore 32300 CPU core contains a 16 entry TLB and that the Wired register contains the capability of indicating up to 64 TLB entries. In programming, the value written to the Wired register must be within the valid range of the number of entries of the current device. For future versions of this core, the RISCore 32300 CPU core implements additional bits.

Bad Virtual Address Register (BadVAddr) (8)

The Bad Virtual Address register (*BadVAddr*) is a read-only register that displays the most recent virtual address that caused one of the following exceptions:

- *Address Error (for example, unaligned access)*
- *TLB Invalid*
- *TLB Modified*
- *TLB Refill*
- *Virtual Coherency Data Access*
- *Virtual Coherency Instruction Fetch*

The processor does not write to the *BadVAddr* register when the *EXL* bit in the *Status* register is set to 1.

Figure 5.10 shows the format of the *BadVAddr* register.

Notes

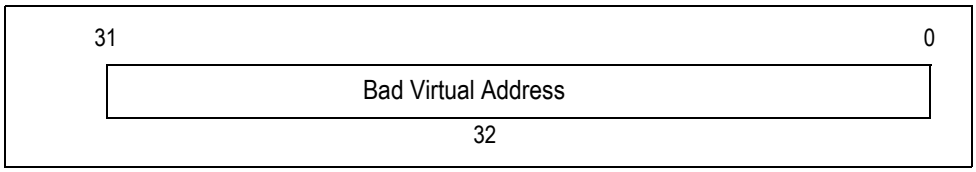


Figure 5.10 Bad Virtual Address Register (BadVAddr) Format

Note: The *BadVAddr* register does not retain information for bus errors, since bus errors are not addressing errors.

EntryHi Register (10)

The *EntryHi* register holds the high-order bits of a TLB entry for TLB read and write operations and is accessed by the TLB Probe, TLB Write Random, TLB Write Indexed, and TLB Read Indexed instructions.

When either a TLB refill, TLB invalid, or TLB modified exception occurs, the *EntryHi* register is loaded with the virtual page number pair (VPN2) and the ASID of the virtual address that did not have a matching TLB entry. Table 5.10 shows the Entry Hi register format and lists the field content descriptions.

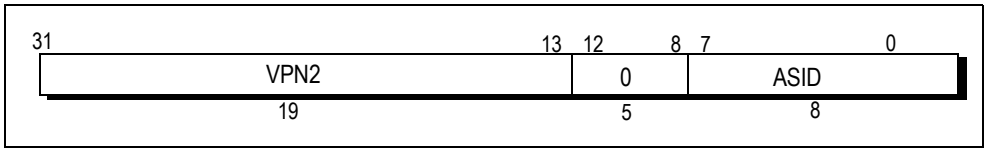


Figure 5.11 EntryHi Register Format

Field	Description
VPN2	Virtual page number divided by two (maps to two pages).
ASID	Address space ID field. An 8-bit field that lets multiple processes share the TLB; each process has a distinct mapping of otherwise identical virtual page numbers.
0	Reserved. Must be written as zeroes, returns zeroes when read.

Table 5.10 EntryHi Register Field Content Descriptions

Kernel/User Operating Modes and Addressing

The RC32300 CPU core supports both the Kernel and User operating modes. The operating system uses Kernel mode for privileged programs; User mode executes non-privileged programs. The CPU enters *Kernel* mode whenever an exception occurs and remains in this mode until the ERET (Exception Return) instruction is executed.

User Mode

The CPU is in User mode when the Status register has the following values:

- *UM bit is 1*
- *EXL bit is 0*
- *ERL bit is 0*

While in user mode, a single, uniform virtual address space of 2 Gbytes is available for the user's program. All references to this address space are mapped by the virtual address mapping mechanism described earlier. The cacheability is controlled by "cache mode" bits in the TLB.

Notes

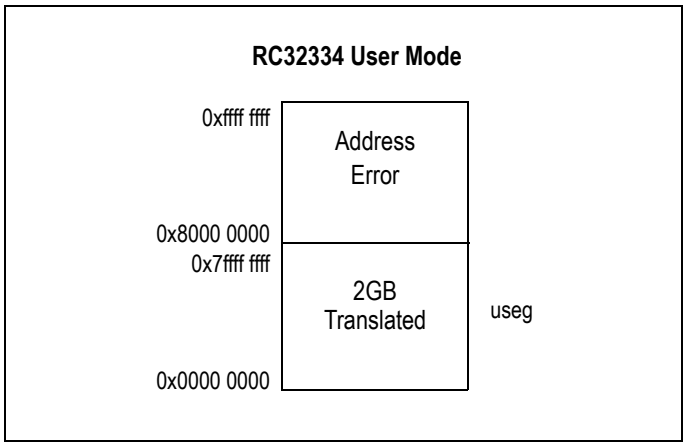


Figure 5.12 Illustration of RC32334 User Mode Address Space

Kernel Mode

The CPU is in Kernel mode when the status register contains any one of the following bit-settings:

- *UM bit is 0*
- *EXL bit is 1*
- *ERL bit is 1*

While in Kernel Mode, the virtual address space is partitioned into the following segments:

- *kuseg*

  This virtual address space is selected if the most significant bit of the virtual address is cleared. This space covers the full 2 GBytes of the current user address space. The virtual address is extended with the contents of the ASID field to form unique virtual addresses.

- *kseg0*

  This virtual address space is selected if the most significant three bits of virtual address are 100_2. References to kseg0 are not "mapped"; the physical address is calculated by subtracting 0x8000_0000 from the virtual address. Cacheability and coherency are controlled by the K0C field of the Configuration Register.

- *kseg1*

  This virtual address space is selected if the most significant three bits of virtual address are 101_2. References to kseg1 are not mapped; the physical address is calculated by subtracting 0xa000_0000 from the virtual address. Caches are always disabled for accesses to this space, and physical memory (or memory-mapped I/O device registers) are accessed directly.

- *kseg2*

  This virtual address space is selected if the most significant 8 bits of virtual address are from $c0_{16}$ to fe_{16}. This space covers the upper 1008 MBytes of kernel virtual address space. The virtual address is extended with the contents of the ASID field to form unique virtual addresses. These addresses are translated to a physical address through the TLB.

- *On-chip/ICE Registers (if the configuration register bit 3 is set to 0)*

  The upper-most 16 MBytes of the virtual address is reserved for memory-mapped on-chip registers and In-Circuit Emulator space. On-chip memory controller and peripheral have their register set mapped into this address space.

  If the configuration register bit 3 is set to 1, this space is considered as kseg2.

Notes

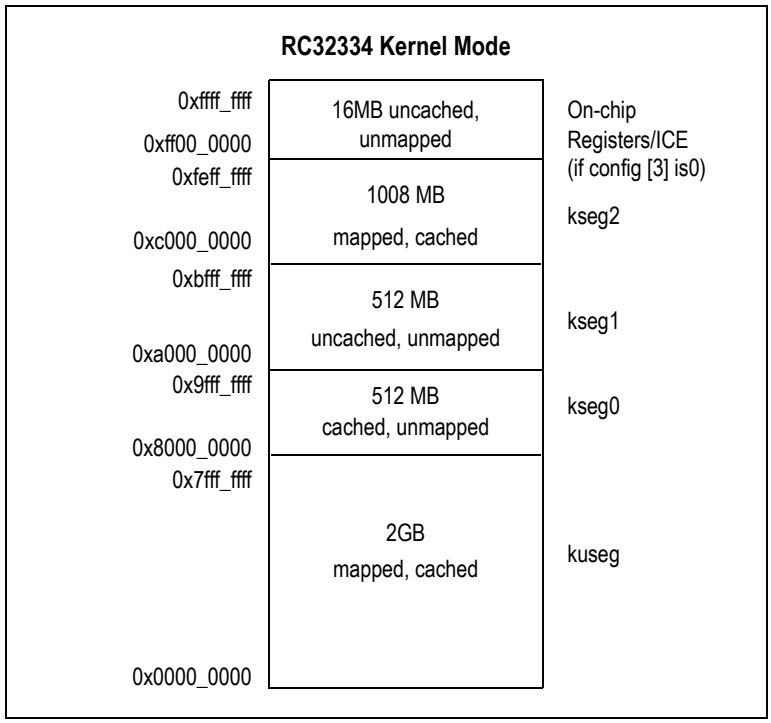


Figure 5.13 Illustration of RC32334 Kernel Mode Address Space

For complete field and content descriptions as well as virtual address locations for the Port Width and Bus Turnaround control registers, refer to Chapter 8 of this manual.

CPU Exception Processing

Notes

Introduction

The CPU exception process begins when the processor receives and detects exceptions from sources such as address translation errors, arithmetic overflows, I/O interrupts, and system calls.

Once an interrupt is detected, the processor suspends the normal instruction sequence and enters Kernel mode (information on system operating modes is located in Chapter 5). The processor then disables interrupts and forces execution of a software exception processor (known as a handler), which is located at a fixed address.

The handler may save the context of the processor—including the program counter contents, the current operating mode (User or Kernel mode), and the interrupt status (enabled or disabled)—so it can be restored when the exception has been serviced.

The RC32300 CPU core supports the following basic exceptions, which are listed from the highest to the lowest priority order:

- *Reset*
- *In-Circuit Emulation*
- *Soft Reset*
- *Nonmaskable Interrupt (NMI)*
- *Address Error caused by Instruction fetch*
- *Watch exception caused by Instruction fetch*
- *Cache Error caused by Instruction fetch*
- *Bus Error caused by Instruction fetch*
- *Integer Overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unusable*
- *Address Error caused by Data access*
- *Cache Error caused by Data access*
- *Watch exception caused by Data access*
- *Bus Error caused by Data access*
- *Interrupt*

Exception Processing Registers

Support for the basic exceptions listed above is implemented through the CP0 exception processing registers, which assist by retaining address, cause and status information.

For example, when an exception occurs, the CPU loads register 14—the Exception Program Counter (EPC) register—with a location from which execution can restart after the exception has been handled. The restart location loaded into the EPC register is either the address of the instruction that caused the exception or the address of the branch instruction immediately preceding the delay slot, if the instruction was executing in a branch delay slot.

A list of basic CP0 registers is given in Table 6.1. Following the table, a brief operational description of each exception register is provided. Those listed as MMU registers are discussed further in Chapter 5, “Memory Management.”

Notes

Number	Register	Description
0 - 8	___	Used for MMU registers. (See Chapter 5 for register descriptions)
9	Count	Timer Count
10	___	Used for MMU. (See Chapter 5 for register descriptions)
11	Compare	Timer Compare
12	Status	Status Register
13	Cause	Cause of last exception
14	EPC	Exception Program Counter
15	PRId	Processor Revision Identifier
16	Config	Configuration register
17	___	Reserved
18	IWatch	Instruction Breakpoint Virtual address
19	DWatch	Data Breakpoint Virtual address
20-21	___	Reserved
22	IEPC	Imprecise Exception Program Counter
23	DEPC	Debug Exception Program Counter
24	Debug	Debug control/status register.
25	—	Reserved
26	ECC	Primary cache Parity
27	CacheErr	Cache Error and Status register
28	TagLo	Cache Tag register
29	___	Reserved
30	ErrorEPC	Error Exception Program Counter
31	___	Reserved

Table 6.1 Basic CP0 Registers

Count Register (9)

The *Count* register is a read/write register that acts as a timer, incrementing at a constant rate—half the maximum instruction issue rate—whether or not an instruction is executed, retired, or any forward progress is made through the pipeline.

This register can be written to for either diagnostic purposes or system initialization; for example, to synchronize processors. Figure 6.1 shows the format of the *Count* register.

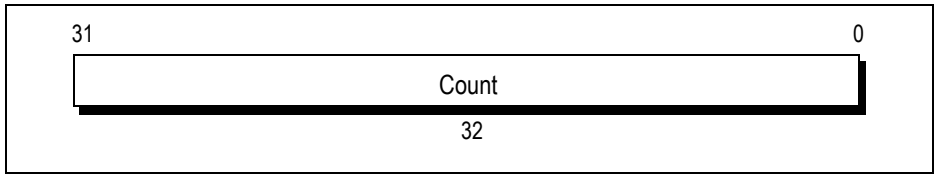


Figure 6.1 Count Register Format

Notes

Compare Register (11)

The *Compare* register acts as a timer (also see the *Count* register), and it maintains a stable value that does not change on its own.

When the value of the *Count* register equals the value of the *Compare* register, interrupt bit *IP(7)* in the *Cause* register is set to initiate a timer interrupt, which causes an interrupt as soon as it's enabled.

Writing a value to the *Compare* register clears the timer interrupt. For diagnostic purposes, the *Compare* register is both a read and write register. However, during normal operations, the *Compare* register is a write only. The format of the compare register is shown in Figure 6.2.

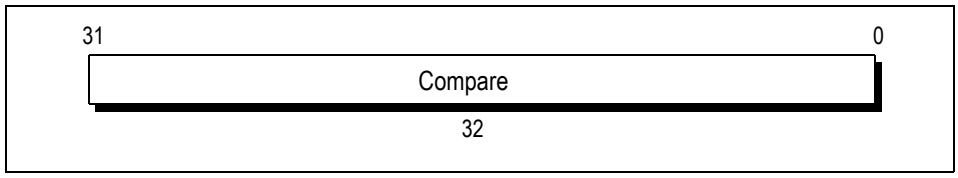


Figure 6.2 Compare Register Format

Status Register (12)

The *Status* register (SR) is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. Figure 6.3 shows the format of the entire register. The following bulleted items provide details on the more important *Status* register fields:

- *The 8-bit Interrupt Mask (IM) field controls the individual enabling of eight interrupt conditions. Interrupts must be generally enabled before they can cause the exception (IE set), and the corresponding bits are set in both the Interrupt Mask field of the Status register and the Interrupt Pending (IP) field of the Cause register (for more information, refer to the Interrupt Pending (IP) field of the Cause register).IM[1:0] are the masks for the two software interrupts and IM[7:2] correspond to Int[5:0].*
- *The 4-bit Coprocessor Usability (CU) field controls the usability of 4 possible coprocessors. Regardless of the CU0 bit setting, CP0 is always usable in Kernel mode. For all other cases, an instruction for or access to an unusable coprocessor causes an exception.*
- *The 9-bit Diagnostic Status (DS) field (Status[24:16]) is used for self-testing and checks the cache and virtual memory system.*
- *The Reverse-Endian (RE) bit, bit 25, reverses the endianness of the machine. At system reset, the processor can be configured as either little-endian or big-endian. This selection is always used in Kernel and Supervisor modes, and also in User mode when the RE bit is 0. Setting the RE bit to 1 inverts the User mode endianness.*

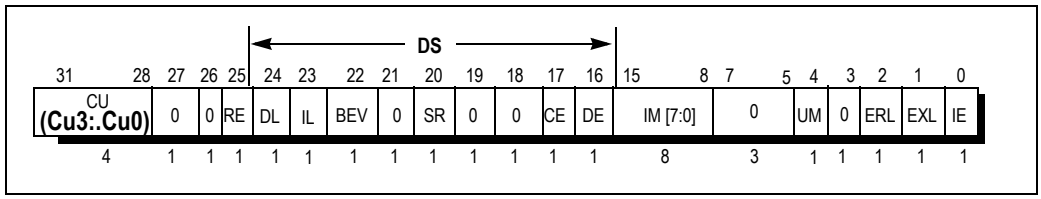


Figure 6.3 Status Register Format

Table 6.2 lists the descriptions for the Status register's fields.

Notes

Field	Description
CU	Controls the usability of each of the four coprocessor unit numbers. CP0 is always usable when in Kernel mode, regardless of the setting of the CU_0 bit. 1 → usable 0 → unusable
RE	Reverse-Endian, valid in User mode.
DL	Data Cache Lock enable. This bit enables the data cache lock function. If this bit is set during Data cache fill, the cache line at that particular set will be locked. Please refer to the “Cache Operation” section for more detail 0 → disable Data cache locking 1 → enable Data cache locking
IL	Instruction Cache Lock enable. This bit enables the instruction cache lock function. If this bit is set during Instruction cache fill, the cache line at that particular set will be locked. Please refer to the “Cache Operation” section for more detail 0 → disable Instruction cache locking 1 → enable Instruction cache locking
BEV	Controls the location of TLB refill and general exception vectors. 0 → normal 1 → bootstrap
SR	1→ Indicates a soft reset or NMI has occurred.
CE	Contents of the ECC register set or modify the check bits of the caches when CE = 1; see description of the *ECC* register.
DE	Specifies that cache parity errors cannot cause exceptions. 0 → parity remains enable 1 → disables parity
0	Reserved. Must be written as zeroes, and returns zeroes when read.
IM	Interrupt Mask: controls the enabling of each of the external, internal, and software interrupts. An interrupt is taken if interrupts are enabled, and the corresponding bits are set in both the *Interrupt Mask* field of the *Status* register and the *Interrupt Pending* field of the *Cause* register. IM[7:2] correspond to interrupts Int[5:0] and IM[1:0] to the software interrupts. 0 → disabled 1 → enabled
UM	User Mode bits 1 → User 0 → Kernel
ERL	Error Level 0 → normal 1 → error
EXL	Exception Level 0 → normal 1 → exception **Note:** When going from 0 to 1, IE should be disabled (0) first. This would be done when preparing to return from the exception handler, such as before executing the ERET instruction.
IE	Interrupt Enable 0 → disable interrupts 1 → enables interrupts

Table 6.2 Status Register Field Descriptions

Notes

Status Register Modes and Access States

Fields of the *Status* register set the modes and access states as described in the following sections:

Interrupts are enabled when all of the following conditions are true:

IE = 1
EXL = 0
ERL = 0

If these conditions are met, the settings of the *IP* bits identify the interrupt.

Note: Setting the IE bit may be delayed by up to 3 cycles. If performing nested interrupts, re-enable the IE bit first.

Data cache locking is enabled when all of the following conditions are true:

DL = 1
EXL = 0
ERL = 0

If these conditions are met, the filled data cache line at the currently selected set will be locked.

Note: Setting the DL bit may be delayed by as many as 3 cycles.

Instruction cache locking is enabled when all of the following conditions are true:

IL = 1
EXL = 0
ERL = 0

If these conditions are met, the filled instruction cache line at the currently selected set will be locked.

Note: Setting the IL bit may be delayed by as much as 3 cycles.

For User and Kernel modes, the following CPU *Status* register bit settings are required:

The processor is in User mode when the User Mode, Exception Level and Error Level bits are set as follows:

UM = 1 AND
EXL = 0 AND
ERL = 0

When the User Mode, Exception Level and Error Level bits are set as follows, the processor is in Kernel mode:

UM = 0 OR
EXL = 1 OR
ERL = 1

Access to the kernel address space is allowed when the processor is in Kernel mode.

Access to the user address space is allowed in any of the three operating modes.

At reset, the contents of the *Status* register are undefined, except for the following bits:

ERL = 1
BEV = 1

The *SR* bit distinguishes between Reset and Soft Reset (Nonmaskable Interrupt [NMI]).

Cause Register (13)

The 32-bit read/write *Cause* register describes the cause of the most recent exception. Figure 6.4 shows the fields of this register, and Table 6.3 describes the contents of the *Cause* register fields. As listed in Table 6.3, a 5-bit exception code (*ExcCode*) indicates the cause of the most recent exception. All bits in the *Cause* register—with the exception of the *IP(1:0)* bits—are read-only. The *IP(1:0)* bits are used for software interrupts.

Notes

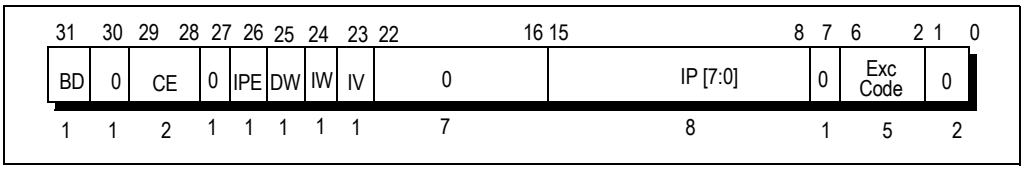


Figure 6.4 Cause Register Format

Field	Description
BD	Indicates whether the last exception taken occurred in a branch delay slot. 1 → delay slot 0 → normal
CE	Coprocessor unit number referenced when a Coprocessor Unusable exception is taken.
DW	On a Watch exception, indicates that the DWatch register matched. On other exceptions this field is undefined.
IW	On a Watch exception, indicates that the IWatch register matched. On other exceptions this field is undefined.
IV	Enable the dedicated interrupt vector. 1 → interrupts use new exception vector (200) 0 → interrupts use dedicated common exception vector (180)
IP	Indicates an interrupt is pending. 1 → interrupt pending 0 → no interrupt
ExcCode	Exception code field
0	Reserved. Must be written as zeroes, and returns zeroes when read.

Table 6.3 Cause Register Field Descriptions

Exception Code Value	Mnemonic	Description
0	Int	Interrupt
1	Mod	TLB modification exception
2	TLBL	TLB exception (load or instruction fetch)
3	TLBS	TLB exception (store)
4	AdEL	Address error exception (load or instruction fetch)
5	AdES	Address error exception (store)
6	IBE	Bus error exception (instruction fetch)
7	DBE	Bus error exception (data reference: load or store)
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic Overflow exception
13	Tr	Trap exception
14	—	Reserved
15:22	—	Reserved
23	Watch	Watch Exception
24:31	—	Reserved

Table 6.4 Cause Register ExcCode Field

Notes

Exception Program Counter (EPC) Register (14)

The Exception Program Counter (*EPC*) is a read/write register that contains the address from which processing resumes after an exception has been serviced.

For synchronous exceptions, the *EPC* register contains either:

- *the virtual address of the instruction that was the direct cause of the exception, or*
- *the virtual address of the immediately preceding branch or jump instruction (when the instruction is in a branch delay slot, and the Branch Delay bit in the Cause register is set).*
- *For an imprecise exception, EPC contains the instruction of the address that recognized the exception and the address at which execution may be resumed.*

When the *EXL* bit in the *Status* register is set to 1, the processor does not write to the *EPC* register. Figure 6.5 shows the format of the *EPC* register.

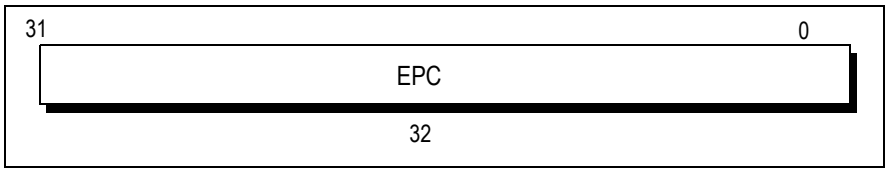


Figure 6.5 EPC Register Format

Processor Revision Identifier (PRId) Register (15)

The 32-bit, read-only *Processor Revision Identifier* (*PRId*) register contains information identifying the implementation and revision level of the CPU and CP0.

Figure 6.6 illustrates the format of the *PRId* register.

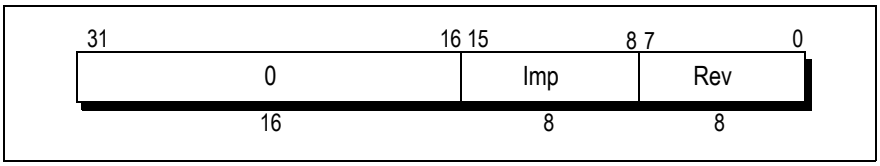


Figure 6.6 PRId Register Format

Table 6.5 describes the contents of the *PRId* register fields.

Field	Description
Imp	Implementation number RC32334: Imp = 0x18
Rev	Revision number. Rw = 0
0	Reserved. Must be written as zeroes, returns zeroes when read.

Table 6.5 PRid Register Field Descriptions

The low-order byte (bits 7:0) of the *PRId* register is interpreted as a revision number, and the high-order byte (bits 15:8) is interpreted as an implementation number.

The implementation number of the RC32334 processor is 0x18. The content of the high-order halfword (bits 31:16) of the register is reserved and will return '0' when read. The revision number is stored as a value in the form *y.x*, where *y* is a major revision number in bits 7:4 and *x* is a minor revision number in bits 3:0.

The revision number can distinguish some chip revisions; however, there is no guarantee that changes to the chip will be reflected in the *PRId* register, or that changes to the revision number necessarily reflects software-visible chip changes. For this reason, these values are not listed and software should not rely on the revision number in the *PRId* register to characterize the chip. Certain attributes, such as cache size, are independent of implementation number.

Notes

Config Register (16)

The *Config* register specifies various configuration options selected on the RC32334 processor.

Some configuration options, as defined by *Config* bits 31:3, are set by the hardware during reset and are included in the *Config* register as read-only status bits for software access. The K0 field is the only read/write field (as indicated by *Config* register bits 2:0) and is controlled by software. On reset, these fields are undefined.

Figure 6.7 shows the format of the *Config* register.

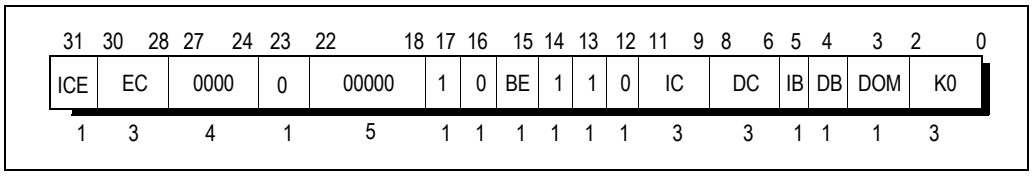


Figure 6.7 Config Register Format

Table 6.6 describes the contents of the *Config* register fields.

Field	Description
ICE	In-Circuit Emulator existence 0 → No ICE hardware connected to the CPU 1 → ICE hardware connected to the CPU These states are determined through an EJTAG Control Register bit.
EC	External Clock: Indicates the relationship of the execution core pipeline clock to the input system clock, as determined at reset: 0 → system clock frequency multiplied by 2 1 → system clock frequency multiplied by 3 2 → system clock frequency multiplied by 4 3 → system clock frequency multiplied by 5 4 → system clock frequency multiplied by 6 5 → system clock frequency multiplied by 7 6 → system clock frequency multiplied by 8 7 Reserved
BE	Big Endian Memory. 0 → Little endian 1 → Big endian The endianness is determined at reset.
IC	Primary I-cache Size (I-cache size = 2^{9+IC} bytes). In the RC32334 controller, this is set to 8 Kbytes (IC = 4)
DC	Primary D-cache Size (D-cache size = 2^{9+DC} bytes). In the RC32334 controller, this is set to 2 Kbytes (DC = 2)
IB	Primary I-cache line size 0 → 16 bytes (4 Words)
DB	Primary D-cache line size 0 → 16 bytes (4 Words)
DOM	Disable On-chip register Mapping 0 → Use the upper-most 16MB of virtual address as memory-mapped on chip register. 1 → Use the upper-most 16MB of virtual address as kseg2.
K0	*kseg0* coherency algorithm (uses same encodings as *EntryLo0* and *EntryLo1* registers, as described in Chapter 5, “Memory Management”.)
Others	Reserved. Returns indicated values when read. Should be written with indicated values.

Table 6.6 Config Register Field Content Descriptions

Notes

IWatch Register (18)

The *IWatch* register is a read/write register that specifies an Instruction virtual address that causes a Watch exception. When $VAddr_{31..2}$ of an instruction fetch matches IvAddr of this register, and the I bit is set, a Watch exception is taken. Matches that occur when EXL=1 or ERL=1 do not take the exception immediately and are instead postponed until both EXL and ERL are cleared. The priority of an IWatch exception is just below an Instruction Address Error exception. Figure 6.8 shows the format of the *IWatch* register.

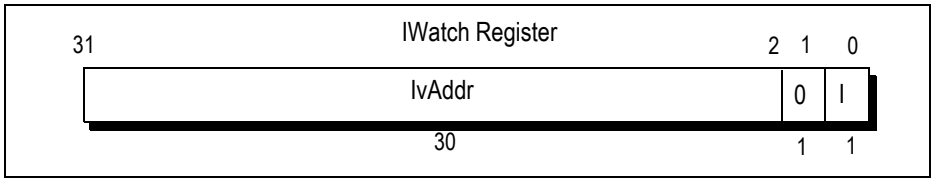


Figure 6.8 IWatch Register Format

Table 6.7 describes the *IWatch* register fields

Field	Description
IvAddr	Instruction virtual address that causes a watch exception [bit 31:2].
I	0 ---> IWatch disable, 1 ---> IWatch enable.
0	reserved for future use.
Note: IWatch.I is cleared on Reset.	

Table 6.7 Watch Register Field Description

DWatch Register (19)

The *DWatch* register is a read/write register that specifies the Data virtual address that caused a Watch exception. When $VAddr_{31..3}$ of a load matches DvAddr of this register and the R bit is set, or when $VAddr_{31..3}$ of a store matches DvAddr of this register and the W bit is set, a Data Watch exception is taken.

Matches that occur when EXL=1 or ERL=1 do not immediately take the exception but are instead postponed until both EXL and ERL are cleared. The priority of a DWatch exception is just below a Data Address Error exception. DWatch exceptions do not occur on CACHE operations. The format of the *DWatch* register is shown in Figure 6.9.

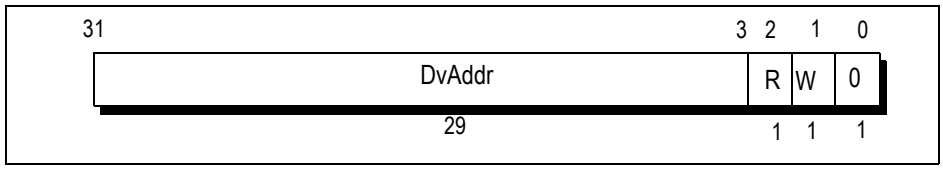


Figure 6.9 DWatch Register Format

Table 6.8 lists the contents of the *DWatch* register's fields.

Field	Description
DvAddr	Data virtual address that causes a watch exception.
R	0 ---> DWatch disable for loads 1 ---> DWatch enable for loads.
W	0 ---> DWatch disable for stores 1 ---> DWatch enable for stores.
0	reserved for future use.
Note: DWatch.R and DWatch.W are cleared on Reset.	

Table 6.8 DWatch Register Field Descriptions

Notes

Debug Exception Program Counter (DebugEPC) Register (23)

This register contains the address of the instruction to resume after the ICE Debug exception is handled.

Debug Register (24)

This register contains status and control bits for the ICE debug operation.

Error Checking and Correcting (ECC) Register (26)

The 8-bit *Error Checking and Correcting* (*ECC*) register reads or writes primary-cache data parity bits for cache initialization, cache diagnostics, or cache error processing. (Tag parity is loaded from and stored to the *TagLo* register). The *ECC* register is loaded by the Index Load Tag CACHE operation. The value of the ECC register is:

- *written into the primary data cache on store instructions (instead of the computed parity) when the CE bit of the Status register is set*
- *substituted for the computed instruction parity for the CACHE operation Fill*

To force a cache parity value, use the *Status CE* bit and the ECC register. Figure 6.10 shows the format of the *ECC* register.

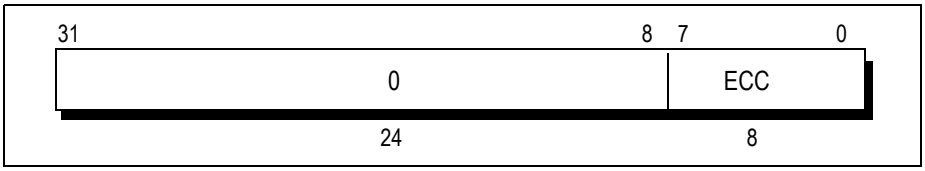


Figure 6.10 ECC Register Format

Table 6.9 describes the contents of the ECC register fields

Field	Description
ECC	An 8-bit field specifying the parity bits read from or written to a primary cache.
0	Reserved. Must be written as zeroes and returns zeroes when read.

Table 6.9 ECC Register Field Descriptions

Cache Error (CacheErr) Register (27)

The 32-bit read-only *CacheErr* register processes parity errors in the primary cache. Parity errors cannot be corrected automatically.

The *CacheErr* register holds cache index and status bits that indicate the source and nature of the error. This register is loaded when a Cache Error exception is asserted. When a read response returns with bad parity this exception is also asserted. Figure 6.11 shows the format of the *CacheErr* register.

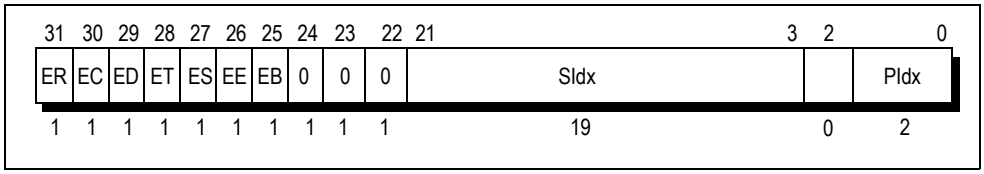


Figure 6.11 CacheErr Register

Table 6.10 provides descriptions on the contents of the *CacheErr* register fields.

Field	Description
ER	Indicates the type of reference as follows: 0 → instruction 1 → data

Table 6.10 Cache Error Register Field Descriptions (Part 1 of 2)

Notes

Field	Description
EC	Cache level of the error 0 → primary
ED	Indicates if a data field error occurred 0 → no error 1 → error
ET	Indicates if a tag field error occurred 0 → no error 1 → error
ES	Reserved
EE	Reserved
EB	Set if a data error occurred in addition to the instruction error (indicated by the remainder of the bits). If so, this requires flushing the data cache after fixing the instruction error. 0 → no additional data error 1 → additional data error
SIdx	Physical address 21:3 of the reference that encountered the error.
PIdx	Virtual address 13:12 of the double word in error. To be used with SIdx to construct a virtual index for the primary caches. Only the lower two bits (bits 1 and 0) are vAddr; the high bit (bit 2) is zero.
0	Reserved. Must be written as zeroes and returns zeroes when read.

Table 6.10 Cache Error Register Field Descriptions (Part 2 of 2)

TagLo Register (28)

The *TagLo* register is a 32-bit read/write register that holds the primary cache tag and parity during cache initialization, cache diagnostics, or cache error processing. The *TagLo* register is written by the CACHE and MTC0 instructions. The *P* field of this register is ignored on Index Store Tag operations. Parity is computed by the store operation.

Figure 6.12 shows the format of the TagLo register, for primary cache operations.

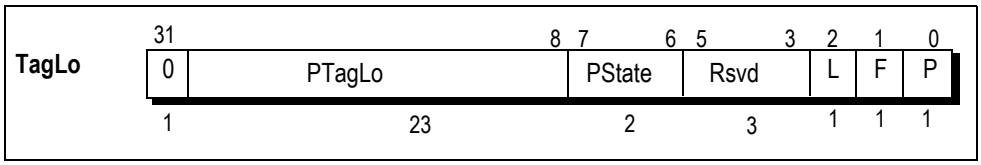


Figure 6.12 TagLo Register Format

Table 6.11 lists the field definitions of the *TagLo* register.

Field	Description
PTagLo	**In the case of Data Cache**, the PTagLo field specifies the physical address bits 31:9. **In the case of Instruction Cache** (8kbytes), the PTagLo field specifies the physical address bits 31:11. The 2 least significant bits are undefined.
PState	Specifies the primary cache state.
P	Specifies the primary tag even parity bit.
F	The FIFO bit used to implement FIFO refill of the cache. For software, there is no particular use of this bit.
Rsvd	Reserved. Must be written as zeroes.
L	Lock bit used to implement cache line lock function.

Table 6.11 TagLo Register Field Descriptions

Notes

Value	Cache State Attribute
0	Invalid
1	Shared
2	Clean Exclusive
3	Dirty Exclusive

Table 6.12 Primary Cache State Values

Error Exception Program Counter (Error EPC) Register (30)

The register is similar to the *EPC* register, except that *ErrorEPC* is used on parity error exceptions (EXL set) and is also used to store the program counter (PC) on Reset, Soft Reset, and nonmaskable interrupt (NMI) exceptions.

The read/write *ErrorEPC* register contains the virtual address at which instruction processing can resume after servicing an error. This address can be:

- *the virtual address of the instruction that caused the exception*
- *the virtual address of the immediately preceding branch or jump instruction, when this address is in a branch delay slot.*

There is no branch delay slot indication for the *ErrorEPC* register.

Figure 6.13 shows the format of the *ErrorEPC* register.

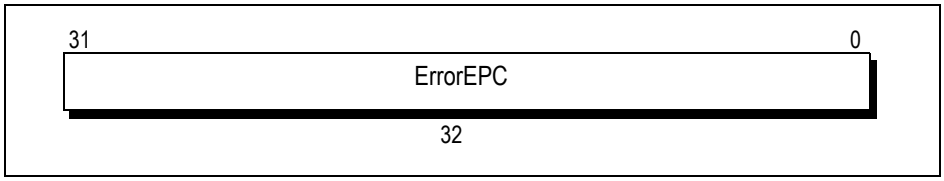


Figure 6.13 ErrorEPC Register

Processor Exceptions

This section describes the processor exceptions: the cause of each exception, its processing by the hardware, and servicing by a handler (software). The types of exception, with exception processing operations, are described in the next section.

Exception Types

This section gives sample exception handler operations for the following exception types:

- *reset*
- *soft reset*
- *nonmaskable interrupt (NMI)*
- *cache error*
- *remaining processor exceptions*

When the *EXL* bit in the *Status* register is 0, either User or Kernel operating mode is specified by the *UM* bits in the *Status* register. When the *EXL* bit or the *ERL* bit is a 1, the processor is in Kernel mode.

When the processor takes an exception, the *EXL* bit is set to 1, which means the system is in Kernel mode. After saving the appropriate state, the exception handler typically resets the *EXL* bit back to 0. When restoring the state and restarting, the handler sets the *EXL* bit back to 1, to inhibit subsequent interrupts. Returning from an exception also resets the *EXL* bit to 0.

In the following sections, sample hardware processes for various exceptions, are shown together with the servicing required by the handler (software).

Notes

General Exception Process

Figure 6.14 shows the process used for exceptions other than Reset, Soft Reset, NMI, and Cache Error.

```
T:  Cause ¨ BD || 0 || CE || 0^12 || Cause_15:8 || 0 || ExcCode || 0^2
    if SR_1 = 0 then                /* system in User mode with no current exception */
      EPC ¨ PC
    endif
    SR ¨ SR_31:2 || 1 || SR0 / + set Exl */
    if SR_22 = 1 then               /* What is the BEV bit setting */
      PC ¨ 0xBFC0 0200 + vector                    /* access to uncached space */
    else
      PC ¨ 0x8000 0000 + vector                    /* access to cached space */
    endif
```

Figure 6.14 General Exception Process

Priority of Exceptions

Although more than one exception can occur for a single instruction, only the exception with the highest priority will be reported. After the highest priority exceptions have been serviced, if lower priority exception conditions remain, they will be signalled and serviced at that time.

The remainder of this chapter describes exceptions—in the order of their priority—as shown in Table 6.13.

Exception Priority			
1	Reset *(highest priority)*	11	Bus error — Instruction fetch
2	Debug (ICE)	12	Integer overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unusable, or Floating-Point Exception
3	Soft Reset	13	Address error — Data access
4	Nonmaskable Interrupt (NMI)	14	TLB refill — Data access
5	Imprecise Bus Error	15	TLB invalid — Data access
6	Address error — Instruction fetch	16	TLB modified — Data write
7	TLB refill — Instruction fetch	17	Cache error — Data access
8	TLB invalid — Instruction fetch	18	Watch -- Data access
9	Watch -- Instruction fetch	19	Bus error — Data access (precise)
10	Cache error — Instruction fetch	20	Interrupt *(lowest priority)*

Table 6.13 Exception Priority Order (highest to lowest)

Generally speaking, the exceptions that will be described in the following sections are handled (“processed”) by hardware; these exceptions are then serviced by software.

Exception Vector Locations

The Reset, Soft Reset, and NMI exceptions are always vectored to location 0xBFC0 0000 (virtual address), corresponding to *kseg1*.

The debug exception for In-Circuit Emulator (ICE) is vectored to location 0xFF20_0200 (virtual address), corresponding to ICE space, if the ICE hardware is connected to the CPU (i.e. Configuration register ICE bit is set). Otherwise, this exception is vectored to location 0xBFC0_0480.

Addresses for all other exceptions are a combination of a *vector offset* and a *base address*. The base address is determined by the *BEV* bit of the *Status* register, as shown in Table 6.16. Table 6.14 lists the vector offset that is added to the base address to create the exception address.

Notes

BEV	Normal Exception Base	Cache Error Base
0	0x8000 0000	0xA000 0000
1	0xBFC0 0200	0xBFC0 0200

Table 6.14 Base Address Vector Offset

As shown in Table 6.15, when *BEV* = 0, the vector base for the Cache Error exception changes from *kseg0* (0x8000 0000) to *kseg1* (0xA000 0000).

When *BEV* = 1, the vector base for the Cache Error exception is 0xBFC00200. This is an uncached and unmapped space, allowing the exception to bypass the cache and TLB.

Exception	RC32334 Processor Vector Offset
TLB refill, EXL = 0	0x000
Cache Error	0x100
Interrupt[1]	0x200
Others	0x180

Table 6.15 List of RC32334 Exception vectors

1. If cause.IV = 1. Otherwise interrupts use general vector offset.

Exception	BEV	EXL	IV	ICE	RC32334 Processor Vector
Reset, Soft Reset, NMI	X	X	X	X	0xBFC0 0000
Debug (ICE)	X	X	X	1	0xFF20 0200
Debug (ICE)	X	X	X	0	0xBFC0 0480
TLB refill	1	0	X	X	0xBFC0 0200
TLB refill	1	1	X	X	0xBFC0 0380
TLB refill	0	0	X	X	0x8000 0000
TLB refill	0	1	X	X	0x8000 0180
Cache Error	1	X	X	X	0xBFC0 0300
Cache Error	0	X	X	X	0xA000 0100
Interrupt	1	X	1	X	0xBFC0 0400
Interrupt	1	X	0	X	0xBFC0 0380
Interrupt	0	X	1	X	0x8000 0200
Interrupt	0	X	0	X	0x8000 0180
Others	1	X	X	X	0xBFC0 0380
Others	0	X	X	X	0x8000 0180
Note: X means don't care					

Table 6.16 RC32334 Exception Vectors

Reset Exception

Cause: The Reset exception occurs when the cpu_coldreset_n signal is asserted and then deasserted.

Processing: The CPU provides a special exception vector for this exception of: 0xBFC0 0000

The Reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to process this exception. It also means the processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

Maskable: No

Notes

The contents of all registers in the CPU are undefined when this exception occurs, except for the following register fields:

- *In the Status register, SR is cleared to 0, and ERL and BEV are set to 1. All other bits are undefined.*
- *The Random register is initialized to the value of its upper bound.*
- *The Wired register is initialized to 0.*
- *Iwatch.I,Dwatch.W and Dwatch.R are cleared.*
- *Some of the Config Register bits are initialized from the boot-time mode stream.*

The Reset exception is serviced by:

- *initializing all processor registers, coprocessor registers, caches, and the memory system*
- *performing diagnostic tests*
- *bootstrapping the operating system*

The Reset exception process is as shown in Figure 6.15.

```
T: undefined
   Random ¨ TLBENTRIES–1
   Wired ¨ 0
   Config <- ICE || EC || EP || 00000000 || BE || 110 || 100 || 010 || 0 || 0 || 0 || 000
   ErrorEPC ¨ PC
   SR ¨ SR31:23 || 1 || 0 || 0 || SR19:3 || 1 || SR1:0 / * ERL¨ 1, BEV ¨ 1 * /
   PC ¨ 0xBFC0 0000
```

Figure 6.15 Process of the Reset Exception

Debug Exception

Cause: The Debug exception occurs either when the ICE Breakpoint signal is asserted from the ICE hardware or when the processor executes the SDBBP instruction.

Processing: The CPU provides a special exception vectors for this exception at:

- *0xFF20 0200 if the ICE hardware is connected to the CPU.*
- *0xBFC0 0480 if the ICE hardware is not connected to the CPU.*

The Debug exception vectors reside in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to process this exception. It also means the processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

Servicing: The Debug exception is serviced by the ICE software, to assist the user in a system level debug.

Maskable: No

Soft Reset Exception

Cause: The Soft Reset exception occurs in response to the Reset* input signal (internal to CPU core), and execution begins at the Reset vector when Reset* is deasserted.

Processing: The Reset exception vector is used for this exception, located within unmapped and uncached address space so that the cache and TLB need not be initialized to process this exception. When a Soft Reset occurs, the *SR* bit of the *Status* register is set to distinguish this exception from a Reset exception.

The primary purpose of the Soft Reset exception is to reinitialize the processor after a fatal error during normal operations. Unlike an NMI, all cache and bus state machines are reset by this exception.

Like Reset, Soft Reset can be used on the processor in any state. The caches, TLB, and normal exception vectors need not be properly initialized.

Notes

When this exception occurs, the contents of all registers are preserved except for:

- *ErrorEPC register, which contains the restart PC*
- *ERL bit of the Status register, which is set to 1*
- *SR bit of the Status register, which is set to 1*
- *BEV bit of the Status register, which is set to 1*

Because the Soft Reset can abort cache and bus operations, cache and memory state is undefined when this exception occurs.

Servicing: The Soft Reset exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing for the Reset exception.

Maskable: No

The Soft Reset and NMI exception processes are as shown in Figure 6.16.


```
T:  ErrorEPC ¨ PC
    SR ¨ SR31:23 || 1 || 0 || 1 || SR19:3 || 1 || SR1:0 /* BEV¨ 1, SR ¨ 1, ERL ¨ 1 */
    PC ¨ 0xBFC0 0000
```

Figure 6.16 Process of the Soft Reset and NMI Exceptions

Nonmaskable Interrupt (NMI) Exception

Cause: The Nonmaskable Interrupt (NMI) exception occurs in response to the asserting edge of the NMI pin. Unlike all other interrupts, this interrupt is not maskable; it occurs regardless of the settings of the *EXL*, *ERL*, and the *IE* bits in the *Status* register.

Processing: The Reset exception vector is used for this exception. This vector is located within unmapped and uncached address space so that the cache and TLB need not be initialized to process an NMI interrupt. When an NMI exception occurs, the *SR* bit of the *Status* register is set to differentiate this exception from a Reset exception. Because an NMI can occur in the midst of another exception, it is not normally possible to continue program execution after servicing an NMI.

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

To terminate a pending read that has hung the best approach is to return a bus error. However, if you wish to use a CPU exception to indicate a hung read, Soft Reset is preferable to NMI.

When this exception occurs, the contents of all registers are preserved except for the following:

- *ErrorEPC register, which contains the restart PC*
- *ERL bit of the Status register, which is set to 1*
- *SR bit of the Status register, which is set to 1*
- *BEV bit of the Status register, which is set to 1*

Servicing: The NMI exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing the system for the Reset exception.

Maskable: No.

Address Error Exception

Cause: The Address Error exception occurs when an attempt is made to execute one of the following:

- *load, fetch, or store a word that is not aligned on a word boundary (except for use of special instruction)*
- *load or store a halfword that is not aligned on a halfword boundary*
- *reference the kernel address space from User mode*

Notes

Processing: The common exception vector is used for the address error exception. If the *AdEL* or *AdES* code in the *Cause* register is set, this indicates how the instruction (shown by the *EPC* register and the *BD* bit in the *Cause* register) caused the exception: through an instruction reference, a load operation, or a store operation.

When this exception occurs, the *BadVAddr* register retains the virtual address that was not properly aligned or had referenced protected address space. The contents of the *VPN* field of the *Context* and *EntryHi* registers are undefined, as are the contents of the *EntryLo* register.

The *EPC* register contains the address of the instruction that caused the exception, unless this instruction is in a branch delay slot. If it is in a branch delay slot, the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set as indication.

Servicing: Typically, the process executing at the time is handed a segmentation violation signal. This error is usually fatal to the process incurring the exception. To resume execution, the *EPC* register or the load/store target address must be altered so that the unaligned reference instruction does not re-execute; this is accomplished by adding a value of 4 to the *EPC* register (*EPC* register + 4) before returning.

If an unaligned reference instruction is in a branch delay slot, interpretation of the branch instruction is required to resume execution.

Maskable: No

TLB Exceptions

This section explains the TLB Exceptions. Three types of TLB exceptions can occur:

- ***TLB Refill*** *occurs when there is no TLB entry that matches an attempted reference to a mapped address space.*
- ***TLB Invalid*** *occurs when a virtual address reference matches a TLB entry that is marked invalid.*
- ***TLB Modified*** *occurs when a store operation virtual address reference to memory matches a TLB entry which is marked valid but is not dirty (the entry is not writable).*

For specifics on the exceptions listed here, refer to the appropriate subsection.

TLB Refill Exception

Cause: The TLB refill exception occurs when there is no TLB entry to match a reference to a mapped address space.

Processing: This exception sets the *TLBL* or *TLBS* code in the *ExcCode* field of the *Cause* register. This code indicates whether the instruction, as shown by the *EPC* register and the *BD* bit in the *Cause* register, caused the miss by an instruction referenced load operation or by a store operation.

When this exception occurs, the *BadVAddr*, *Context*, and *EntryHi* registers hold the virtual address that failed the address translation. The *EntryHi* register also contains the ASID from which the translation fault occurred. The *Random* register normally suggests a valid location in which to place the replacement TLB entry.

The contents of the *EntryLo* registers are undefined. The *EPC* register contains the address of the instruction that caused the exception, unless this instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: To service this exception, the content of the *Context* register is used as a virtual address to fetch memory locations containing the physical page frame and access control bits for a pair of TLB entries. The two entries are placed into the E*ntryLo0/EntryLo1* register; the *EntryHi* and *EntryLo* registers are written into the TLB, typically with a TLBWR instruction.

It is possible that the virtual address used to obtain the physical address and access control information is on a page that is not resident in the TLB. This condition is processed by allowing a TLB refill exception in the TLB refill handler. This second exception goes to the common exception vector because the *EXL* bit of the *Status* register is set.

Maskable: No.

Notes

TLB Invalid Exception

Cause: The TLB invalid exception occurs when a virtual address reference matches a TLB entry that is marked invalid (TLB valid bit cleared).

Processing: The common exception vector is used for this exception. The *TLBL* or *TLBS* code in the *ExcCode* field of the *Cause* register is set, which indicates whether the instruction—shown by the *EPC* register and *BD* bit in the *Cause* register—caused the miss by an instruction referenced load operation or by a store operation.

When this exception occurs, the *BadVAddr*, *Context*, and *EntryHi* registers contain the virtual address that failed address translation. The *EntryHi* register also contains the ASID from which the translation fault occurred. The *Random* register normally contains a valid location in which to put the replacement TLB entry. The contents of the *EntryLo* registers are undefined.

The *EPC* register contains the address of the instruction that caused the exception unless this instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: A TLB entry is typically marked invalid when one of the following is true:

- *a virtual address does not exist*
- *the virtual address exists, but is not in main memory (a page fault)*
- *a trap is desired on any reference to the page (for example, to maintain a reference bit or during debug)*

After servicing the cause of a TLB Invalid exception, the TLB entry is located with TLBP (TLB Probe), and replaced by an entry with that entry's *Valid* bit set.

Maskable: No.

TLB Modified Exception

Cause: The TLB modified exception occurs when a store operation virtual address reference to memory matches a TLB entry that is marked valid but is not dirty and therefore is not writable.

Processing: The common exception vector is used for this exception, and the *Mod* code in the *Cause* register is set. When the TLB Modified Exception occurs, the *BadVAddr*, *Context*, and *EntryHi* registers contain the virtual address that failed address translation. The *EntryHi* register also contains the ASID from which the translation fault occurred. The contents of the *EntryLo* registers are undefined.

The *EPC* register contains the address of the instruction that caused the exception unless that instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: The kernel uses the failed virtual address or virtual page number to identify the corresponding access control information. The page identified may or may not permit write accesses; if writes are not permitted, a write protection violation occurs.

If write accesses are permitted, the page frame is marked dirty/writable by the kernel in its own data structures. The TLBP instruction places the index of the TLB entry that must be altered into the *Index* register. The *EntryLo* register is loaded with a word containing the physical page frame and access control bits (with the *D* bit set), and the *EntryHi* and *EntryLo* registers are written into the TLB.

Maskable: No

Cache Error Exception

Cause: The Cache Error exception occurs when a primary cache parity error is detected.

Processing: The processor sets the *ERL* bit in the *Status* register, saves the exception restart address in *ErrorEPC* register, and then transfers to a special vector in uncached space:

- *If the BEV bit = 0, the vector is 0xA000 0100.*
- *If the BEV bit = 1, the vector is 0xBFC0 0300.*

Notes

No other registers are changed.

Servicing: All errors should be logged. To correct cache parity errors, the system uses the CACHE instruction to invalidate the cache block, overwrites the old data through a cache miss, and resumes execution with an ERET. Other errors are not correctable and are likely to be fatal to the current process.

Maskable: Yes, by the DE bit of the Status register.

The Cache Error exception process is as shown in Figure 6.17.

```
T:  ErrorEPC ¨ PC
    CacheErr ¨ ER || EC || ED || ET || ES || EE || EB || 0^25
    SR ¨ SR31:3 || 1 ||SR1:0                          /* Set ERL */
    if SR22 = 1 then          /* What is the BEV bit setting */
       PC ¨ 0xBFC0 0200 + 0x100                       /* access boot-PROM area */
    else
       PC ¨ 0xA000 0000 + 0x100                       /* access main memory area */
    endif
```

Figure 6.17 Process of the Cache Error Exception

Bus Error Exception

Cause: A Bus Error exception is raised by board-level circuitry for events such as bus time-out, backplane bus parity errors, and invalid physical memory addresses or access types. A Bus Error exception will occur only when a cache miss refill or uncached reference occurs synchronously. A Bus Error exception resulting from a buffered write transaction must be reported using the general interrupt mechanism.

Processing: The common interrupt vector is used for a Bus Error exception. The *IBE* or *DBE* code in the *ExcCode* field of the *Cause* register is set, signifying whether the instruction (as indicated by the *EPC* register and *BD* bit in the *Cause* register) caused the exception by an instruction referenced load operation or store operation.

The *EPC* register contains the address of the instruction that caused the exception, unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: The physical address at which the fault occurred can be computed from information available in the CP0 registers.

If the *IBE* code in the *Cause* register is set (indicating an instruction fetch reference), the virtual address is contained in the *EPC* register. If the *DBE* code is set (indicating a load or store reference), then the instruction that caused the exception is located at the virtual address contained in the *EPC* register (or 4+ the contents of the *EPC* register if the *BD* bit of the *Cause* register is set).

> **Note:** The IPE bit should be checked first. If this bit is set, refer to the servicing section for the Imprecise Bus Error Exception.

The virtual address of the load and store reference can then be obtained by interpreting the instruction. The physical address can be obtained by using the TLBP instruction and reading the *EntryLo* register to compute the physical page number. The process that is executing at the time of this exception is handed a bus error signal, which is usually fatal.

Maskable: No.

Integer Overflow Exception

Cause: An Integer Overflow exception occurs when an ADD, ADDI, SUB[1], or instruction results in a 2's complement overflow.

Processing: The common exception vector is used for this exception, and the *OV* code in the *Cause* register is set.

1. See Appendix A for instruction description.

Notes

The *EPC* register contains the address of the instruction that caused the exception unless the instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: The process executing at the time of the exception is handed a floating-point exception/ integer overflow signal. This error is usually fatal to the current process.

Maskable: No.

Trap Exception

Cause: The Trap exception occurs when a TGE, TGEU, TLT, TLTU, TEQ, TNE, TGEI, TGEUI, TLTI, TLTUI, TEQI, or TNEI[1] instruction results in a TRUE condition.

Processing: The common exception vector is used for this exception, and the *Tr* code in the *Cause* register is set.

The *EPC* register contains the address of the instruction causing the exception unless the instruction is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction and the *BD* bit of the *Cause* register is set.

Servicing: The process executing at the time of a Trap exception is handed a floating-point exception/ integer overflow signal. This error is usually fatal.

Maskable: No.

System Call Exception

Cause: The execution of the SYSCALL instruction causes a System Call exception to occur.

Processing: The common exception vector is used for this exception, and the *Sys* code in the *Cause* register is set.

The *EPC* register contains the address of the SYSCALL instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

If the SYSCALL instruction is in a branch delay slot, the *BD* bit of the *Status* register is set; otherwise, this bit is cleared.

Servicing: When this exception occurs, control is transferred to the applicable system routine. To resume execution, the *EPC* register must be altered so that the SYSCALL instruction does not re-execute; this is accomplished by adding a value of 4 to the *EPC* register (*EPC* register + 4) before returning. If a SYSCALL instruction is in a branch delay slot, a more complicated algorithm, beyond the scope of this description, may be required.

Maskable: No.

Breakpoint Exception

Cause: A Breakpoint exception occurs when an attempt is made to execute the BREAK instruction.

Processing: The common exception vector is used for this exception, and the *Bp* code in the *Cause* register is set. The *EPC* register contains the address of the BREAK instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction. If the BREAK instruction is in a branch delay slot, the *BD* bit of the *Status* register is set, otherwise the bit is cleared.

Servicing: When the Breakpoint exception occurs, control is transferred to the applicable system routine. Additional distinctions can be made by analyzing the unused bits of the BREAK instruction (bits 25:6), and loading the contents of the instruction whose address the *EPC* register contains. A value of 4 must be added to the contents of the *EPC* register *(EPC* register + 4) to locate the instruction if it resides in a branch delay slot.

To resume execution, the *EPC* register must be altered so that the BREAK instruction does not re-execute. This is accomplished by adding a value of 4 to the *EPC* register (*EPC* register + 4) before

Notes

returning. If a BREAK instruction is in a branch delay slot, interpretation of the branch instruction is required to resume execution.

Maskable: No.

Reserved Instruction Exception

Cause: The Reserved Instruction exception occurs when one of the following conditions occurs:

- *an attempt is made to execute an instruction with an undefined major opcode (bits 31:26)*
- *an attempt is made to execute a SPECIAL instruction with an undefined minor opcode (bits 5:0)*
- *an attempt is made to execute a REGIMM instruction with an undefined minor opcode (bits 20:16)*
- *an attempt is made to execute a 64-bit operation*

Processing: The common exception vector is used for this exception, and the *RI* code in the *Cause* register is set. The *EPC* register contains the address of the reserved instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

Servicing: No instructions in the RC32300 CPU core are interpreted. The process executing at the time of this exception is handed an illegal instruction/reserved operand fault signal. This error is usually fatal.

Maskable: No.

Coprocessor Unusable Exception

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

- *a corresponding coprocessor unit that has not been marked usable, or*
- *CP0 instructions, when the unit has not been marked usable and the process executes in User mode.*

Processing: The common exception vector is used for this exception, and the *CPU* code in the *Cause* register is set. The contents of the *Coprocessor Usage Error* field of the coprocessor *Control* register indicate which of the four coprocessors was referenced. The *EPC* register contains the address of the unusable coprocessor instruction unless it is in a branch delay slot, in which case the *EPC* register contains the address of the preceding branch instruction.

Servicing: The coprocessor unit to which an attempted reference was made is identified by the Coprocessor Usage Error field, which results in one of the following situations:

- *If the process is entitled access to the coprocessor, the coprocessor is marked usable and the corresponding user state is restored to the coprocessor.*
- *If the process is entitled access to the coprocessor, but the coprocessor does not exist or has failed, interpretation of the coprocessor instruction is possible.*
- *If the BD bit is set in the Cause register, the branch instruction must be interpreted; then the coprocessor instruction can be emulated and execution resumed with the EPC register advanced past the coprocessor instruction.*
- *If the process is not entitled access to the coprocessor, the process executing at the time is handed an illegal instruction/privileged instruction fault signal. This error is usually fatal.*

Maskable: No.

Interrupt Exception

Cause: The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts is dependent upon the specific system implementation.

Processing: The RC32334 may use the common exception vector or a dedicated vector for this exception, determined by the *Cause* Register *IV* bit. The *Int* code in the *Cause* register is set. The *IP* field of the *Cause* register indicates current interrupt requests. It is possible that more than one of the bits can be simultaneously set (or even *no* bits may be set if the interrupt is asserted and then deasserted before this register is read).

Notes

Servicing: If the interrupt is caused by one of the two software-generated exceptions (*SW1* or *SW0*), the interrupt condition is cleared by setting the corresponding *Cause* register bit to 0. If the interrupt is hardware-generated, the interrupt condition is cleared by correcting the condition causing the interrupt pin to be asserted.

Maskable: Yes. Each of the eight interrupts can be masked by clearing the corresponding bit in the *IntMask* field of the *Status* register, and all of the eight interrupts can be masked at once by clearing the *IE* bit of the *Status* register.

> **Note:** Due to the write buffer, a store to an external device will not necessarily occur until after completion of other instructions in the pipeline. Thus, the user must ensure that the store occurs before the return from exception (ERET) instruction is executed; otherwise, the interrupt may be serviced again, although there should be no interrupt pending. The Sync instruction can be used to achieve this.

DWatch Exception

Cause: DWatch is a read-write register that specifies a data virtual address that causes a Watch exception. This exception occurs either when the program does a load and the target address matches DWatch and DWatch.R is set or when the program does a store and the target address matches DWatch and DWatch.W is set.

Processing: The common exception vector is used for this exception. The Watch code of the *Cause* register is set with the *DW* bit set.

Servicing: This exception is typically used during system debug. Servicing is system-specific.

Maskable: No. Enabled or disabled through bits in the DWatch register (19). Refer to Table 6.8 for settings and descriptions.

IWatch Exception

Cause: IWatch is a read-write register that specifies an instruction virtual address that causes a Watch exception. The exception occurs when the program address matches the IWatch Register, and IWatch.I is set.

Processing: The common exception vector is used for this exception. The Watch code of the *Cause* register is set with the *IW* bit set.

Servicing: Typically, this exception is used during system debug. Servicing is system-specific.

Maskable: No. Enabled or disabled through bits in the IWatch register (18). Refer to Table 6.7 for settings and descriptions.

Exception Handling and Servicing Flowcharts

The remainder of this chapter contains flowcharts for the exceptions described in Table 6.13 as well as guidelines for their handlers.

Figure	Description
Figure 6.18 Figure 6.19	General exceptions and their exception handler (HW) General exceptions and their exception handler (SW)
Figure 6.20 Figure 6.21	TLB miss exception and their exception handler (HW) TLB refill exception servicing guideline (SW)
Figure 6.22	Cache error exception and its handler
Figure 6.23	Reset, soft reset and NMI exceptions, and a guideline to their handler.

Table 6.17 List of Exception Handling Flowchart Types

In general, exceptions are handled by hardware (HW) and serviced by software (SW).

Notes

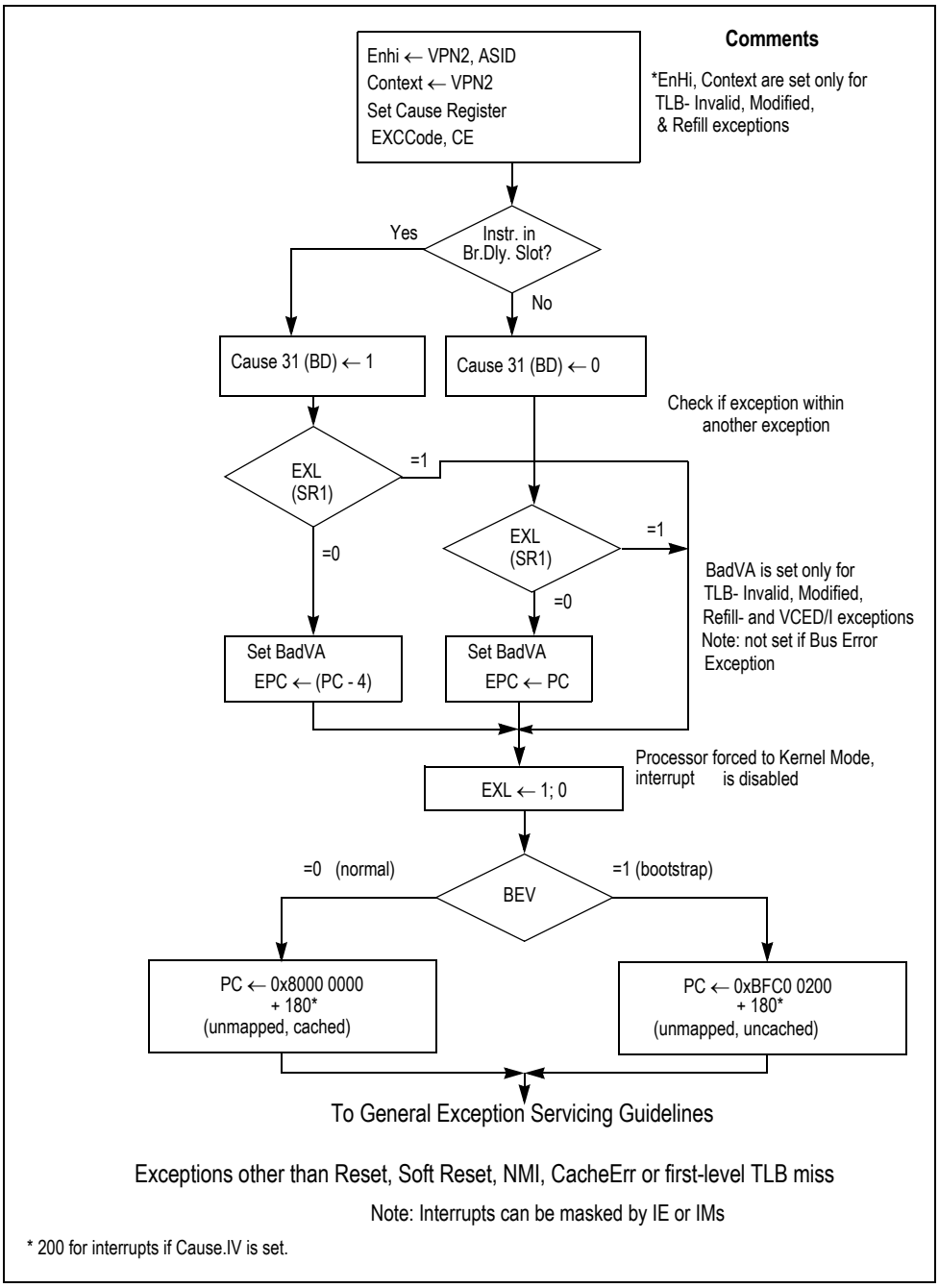


Figure 6.18 General Exception Handling (HW)

Notes

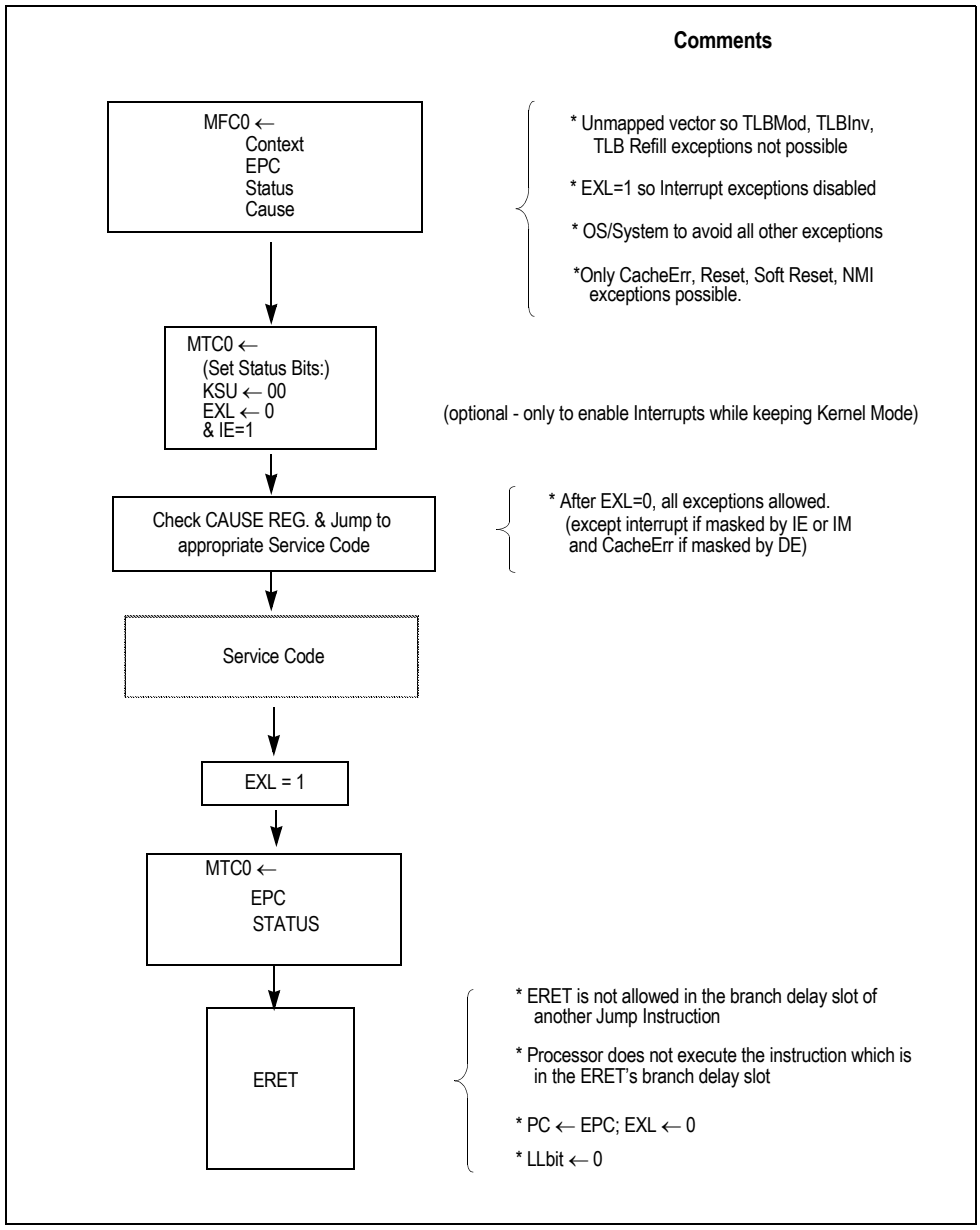


Figure 6.19 General Exception Servicing Guideline (SW)

Notes

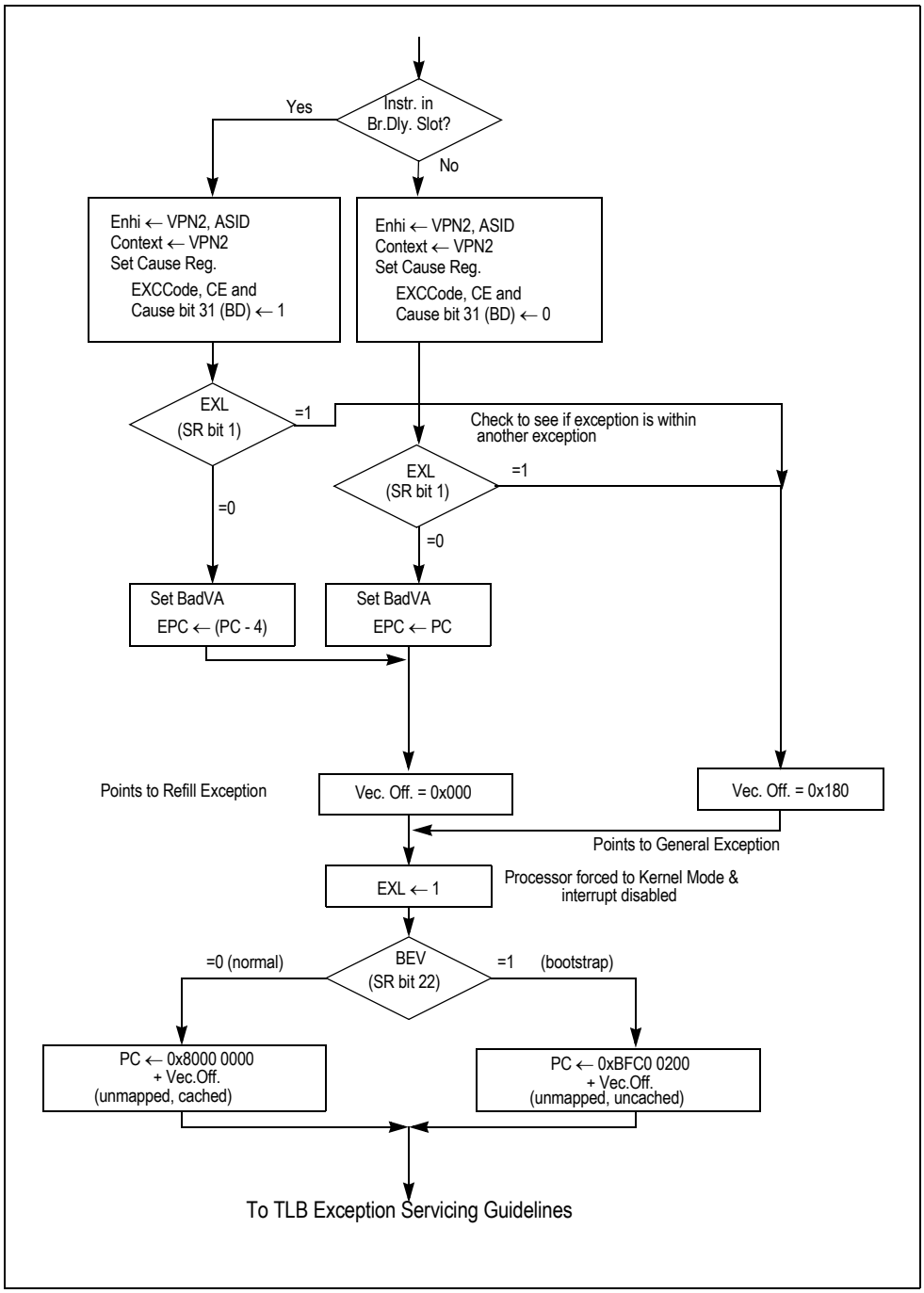


Figure 6.20 TLB Refill Exception Handling (HW)

Notes

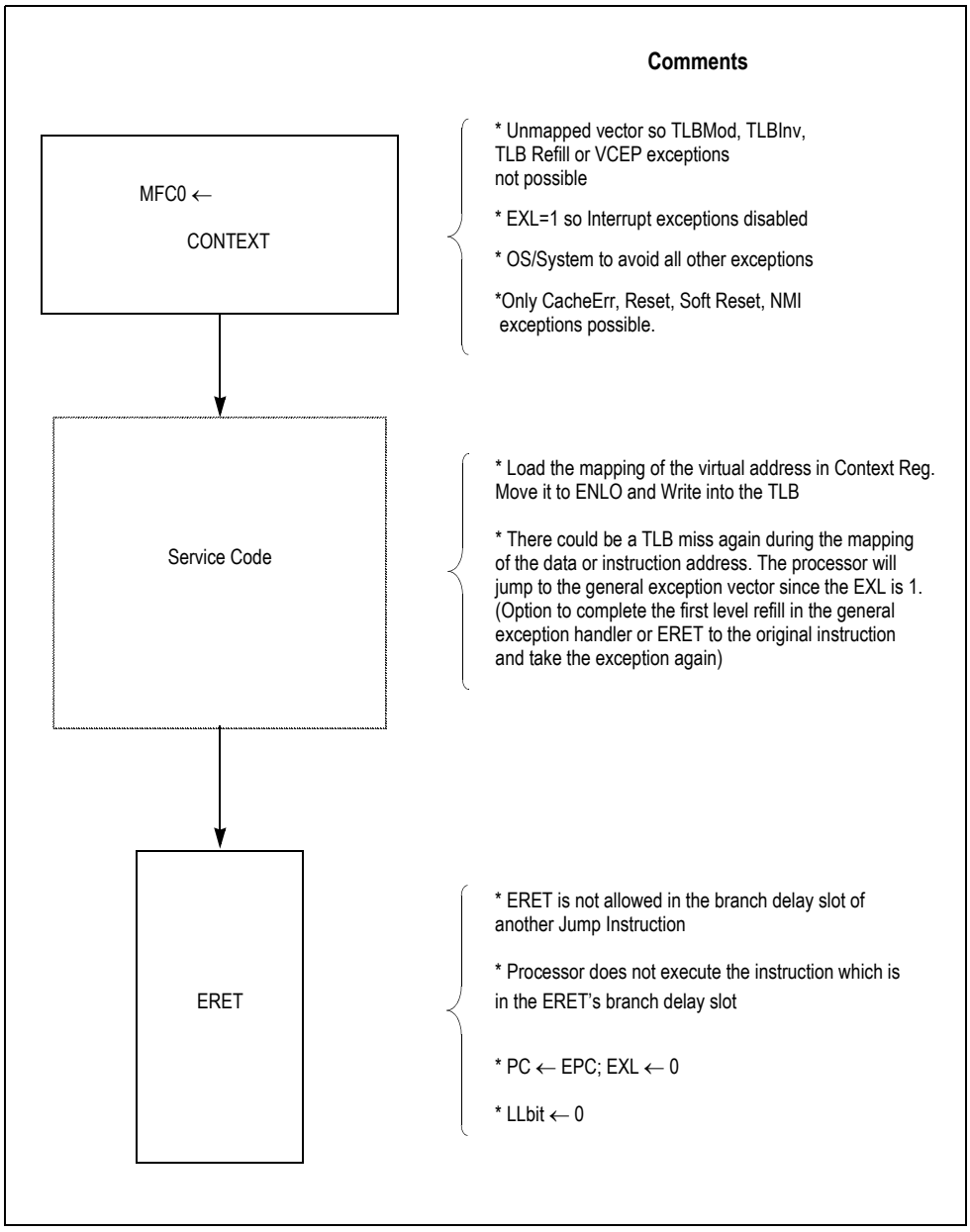


Figure 6.21 TLB Refill Exception Servicing Guideline (SW)

Notes

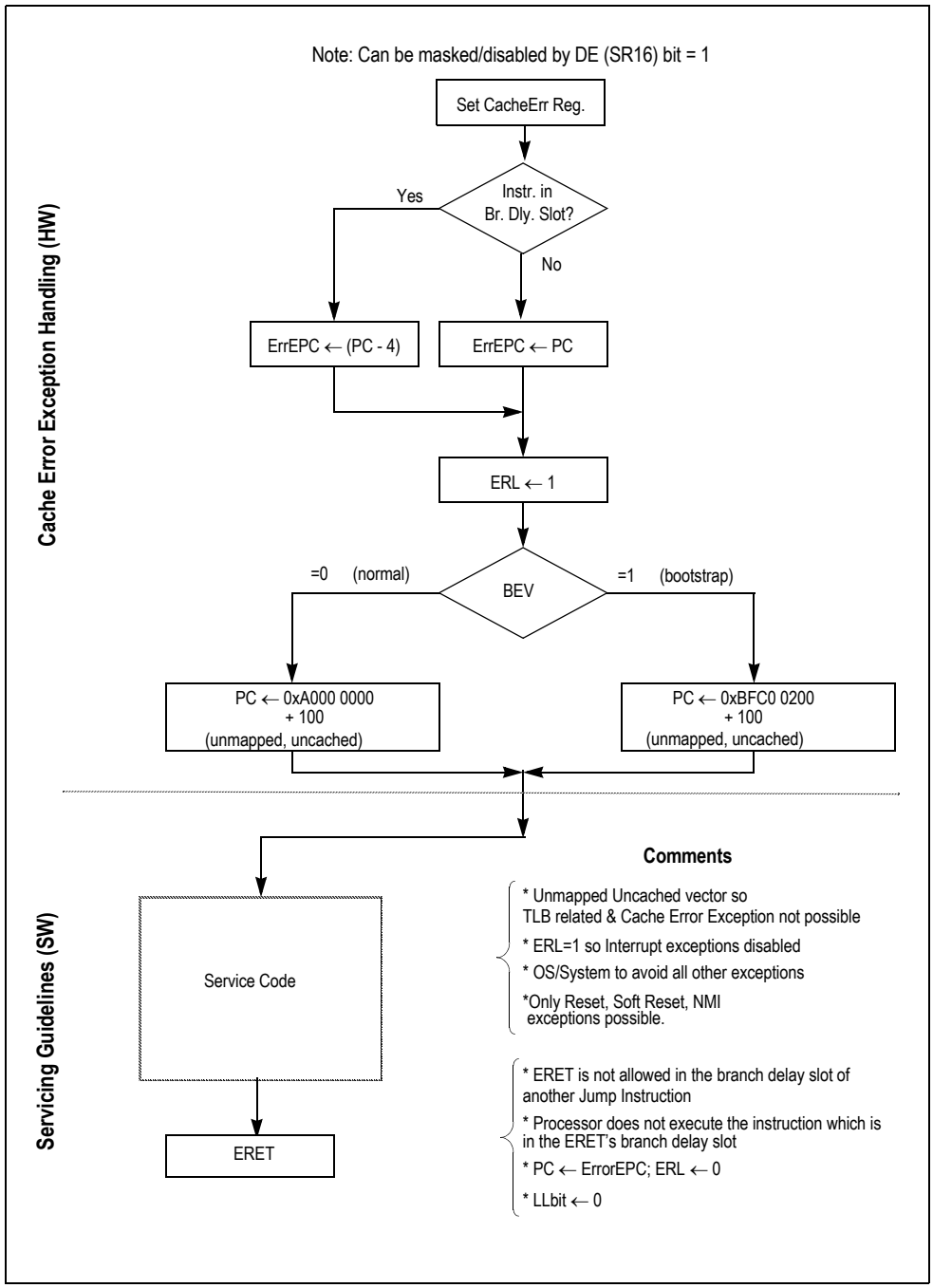


Figure 6.22 Cache Error Exception Handling (HW) and Servicing Guidelines (SW)

Notes

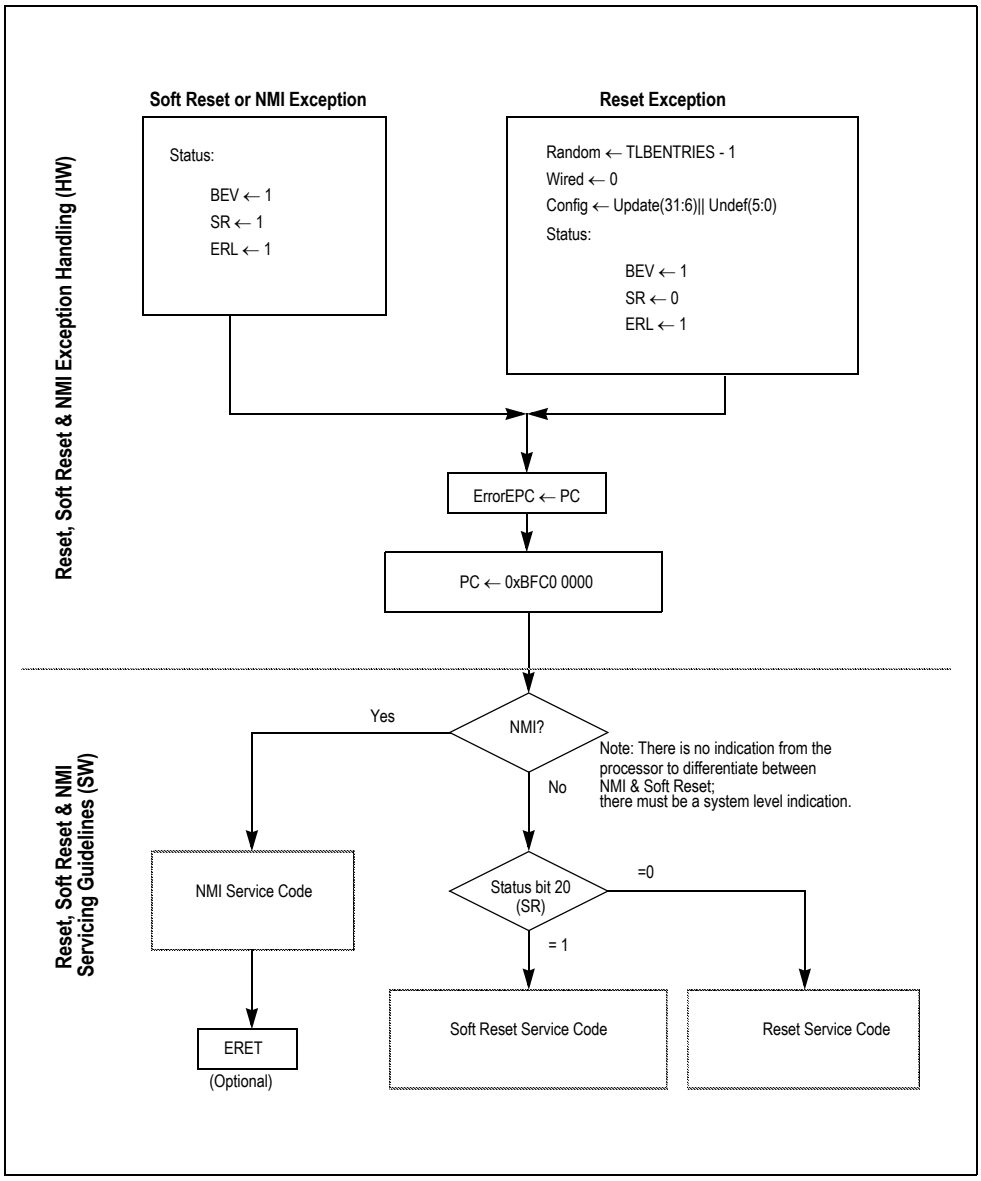


Figure 6.23 Reset, Soft Reset & NMI Exception Handling (HW) and Servicing Guidelines (SW)

Cache Organization, Operation, and Coherency

Notes

Introduction

Caches are small, high speed memories used to buffer the central processing unit from slower, larger storage devices such as those found in main memory. Caches are used to store the data or instructions that a program is currently using while the majority of the data remains in the slower memory, thus providing quick, temporary storage.

In the logical memory hierarchy, caches reside between the CPU and main memory. The increased memory access speed made possible through caches is usually transparent to the programmer.

Each functional block shown in Figure 7.1 has the capacity to hold more data than the block above it. For example, physical main memory has a larger capacity than the primary cache. However, each functional block requires longer access times than any block above it; therefore, it takes longer to access data in main memory than in the CPU on-chip registers.

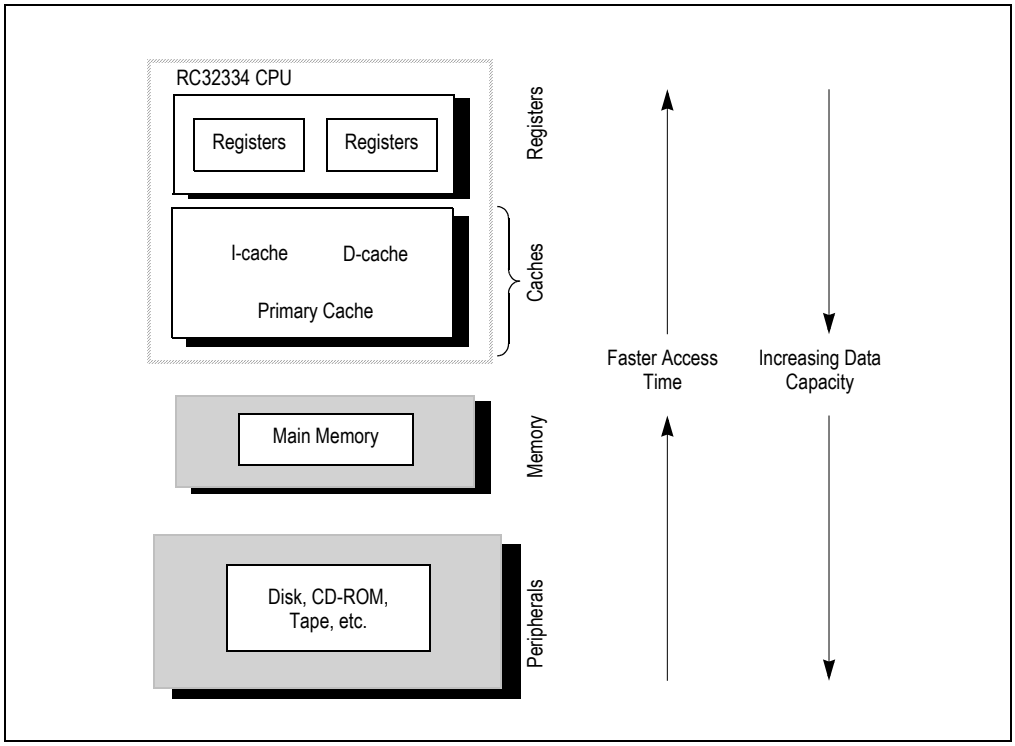


Figure 7.1 Logical Hierarchy of Memory

Cache Operation Overview

To support high-performance RISC designs, the primary cache is made up of an Instruction cache (holds instructions) and a Data cache (holds data). This arrangement allows the processor simultaneous access to both instructions and data, thereby doubling the effective cache-memory bandwidth.

In general, during cache operations, the processor accesses cache-resident instructions/data when the on-chip cache controller detects valid information in the cache by an address match. Figure 7.4 shows the primary cache lookup sequence.

If valid instruction or data is present, the processor retrieves it from cache memory and is then known as a primary-cache *hit.* If the instruction/data is not present, a cache *miss* has occurred. The cache line must then be retrieved from slower main memory.

Notes

For a cache hit, the processor retrieves the instruction/data from the (high-speed) primary cache and the operation continues. In the case of a cache miss, the processor can restart the pipeline after the first doubleword is retrieved (the one at the miss address) and continues the cache line refill in parallel.

It is possible for the same data to simultaneously be in main memory and primary cache. The data is kept consistent through the use of either a write-back or a write-through methodology. For a write-back cache, the modified data is not written back to memory until the cache line is replaced. In a write-through cache, the data is written to memory as the cached data is modified (with a possible delay due to the write buffer).

RC32334 Cache Description

Details of the RC32334's cache memory are provided in the remainder of this chapter. Throughout this text, the following terminology will be used:

- *The primary cache may also be referred to as the P-cache*
- *The primary data cache may also be referred to as the D-cache*
- *The primary instruction cache may also be referred to as the I-cache.*

These terms will also be used interchangeably throughout the manual.

RC32334 Cache Attributes

Table 7.1 highlights the user attributes of the RC32334 caches.

Attribute	Instruction	Data
size	8kB	2kB
organization	2-way set-associative	2-way set associative
line size	16 Bytes	16 Bytes
read unit	32-bits	32-bits
write policy	n.a.	write-back or write-through, as specified in CP0.
line transfer order	sub-block order	sub-block order
miss restart after transfer of	entire line	miss word
Cache-locking	per line	per line
parity	per-word	per-byte

Table 7.1 RC32334 Cache Attributes

Cache Organization and Accessibility

This section describes the organization of the primary cache, including the manner in which it is mapped, the addressing used to index the cache, and composition of the cache lines. The primary instruction and data caches are indexed with a virtual address (VA).[1]

Organization of the Primary Instruction Cache (I-Cache)

Each line of primary I-cache data (although the field actually contains an instruction, it is referred to as data to distinguish it from the tag field) has an associated 25-bit tag that contains a 21-bit physical address, a single valid bit, a single parity bit, a lock bit, and the FIFO replacement bit. Word parity is used on I-cache data.

1. Because the size of one set of primary caches is 8KB for ICache and 2KB for DCache, the virtual offset equals the physical offset. Logically, however, the cache index is pre-translation and thus considered virtual.

Notes

The primary I-cache of the RC32334 processor has the following characteristics:

- *Two-way set associative*
- *Indexed with a virtual address*
- *Checked with a physical tag*
- *Organized with 4-word (16-byte) cache line*
- *Lockable on a per-line basis.*

Figure 7.2 shows the format of a primary I-cache register, and Table 7.2 lists field content descriptions.

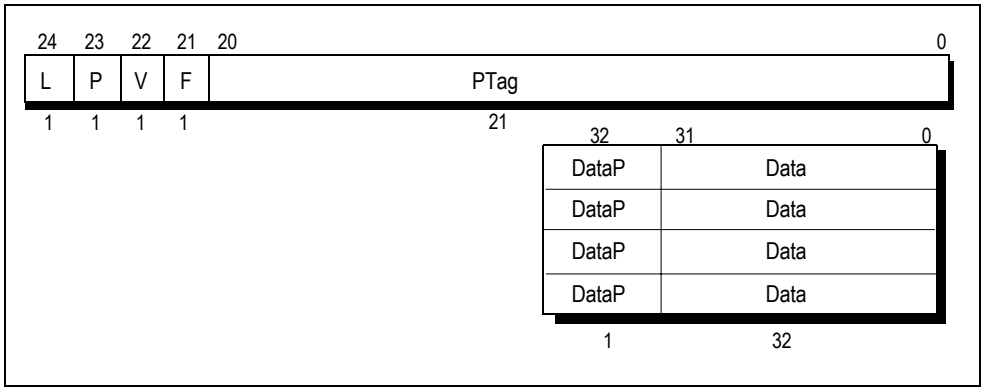


Figure 7.2 Primary I-Cache Line Format

Field	Description
PTag	Physical tag (bits 31:11 of the physical address)
F	FIFO Replacement Bit. Complemented on refill
V	Valid bit
P	Even parity for the PTag and V fields
L	Lock bit
DataP	Even parity; 1 parity bit per word of data
Data	Cache data

Table 7.2 Primary I-Cache Line Field Descriptions

Note: The Physical tag field contains 21 bits (bit [31:11]) of the physical address to support the smaller I-cache size of 4KB (2KB per set) in the future. For the current version of 3200 core with 8KB of I-cache, just bits [31:12] are valid, bit 11 is ignored.

Organization of the Primary Data Cache (D-Cache)

Each line of primary D-cache data has an associated 30-bit tag that contains a 23-bit physical address, 2-bit cache line state, a write-back bit, a parity bit for the physical address and cache state fields, a parity bit for the write-back bit, the FIFO replacement bit, and a lock bit.

The primary D-cache of the RC32334 processor has the following characteristics:

- *Write-back or write-through on a per-page basis*
- *Two-way set associative*
- *Indexed with a virtual address*
- *Checked with a physical tag*
- *Organized with 4-word (16-byte) cache line*
- *Lockable on a per-line basis.*

Notes

Figure 7.3 shows the format of a primary D-cache line; Table 7.3 provides the field content descriptions.

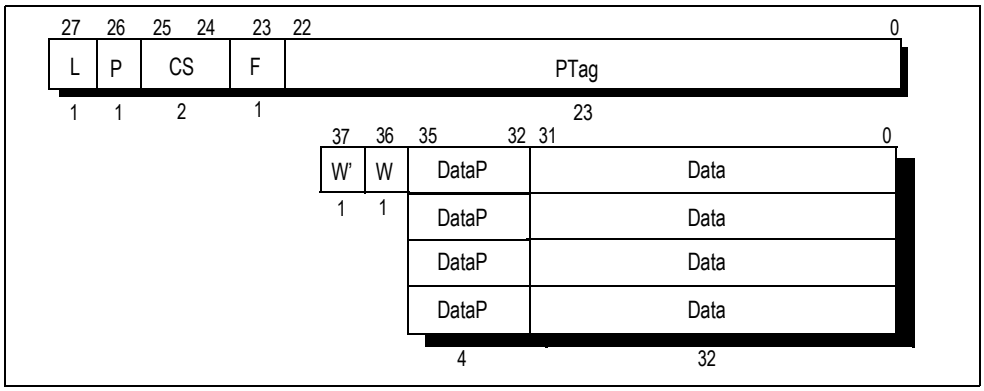


Figure 7.3 Primary D-Cache Line Format

Field	Description
PTag	Physical tag (bits 31:9 of the physical address)
F	FIFO Replacement Bit
CS	Primary cache state: 0 = Invalid, 1 = Shared, 2 = Clean Exclusive, 3 = Dirty Exclusive
P	Even parity for the PTag and CS fields
L	Lock bit
W	Write-back bit (set if cache line has been written)
W	Write-back bit (set if cache line has been written)
DataP	Even parity for the data; 1-bit per byte
Data	Cache data

Table 7.3 Primary D-Cache Line Field Description

Note: The physical tag field contains 23 bits (bits [31:9]) of the physical address to support the smallest D-cache size of 1KB (512B per set) in the future. For the current version of 3200 core with 2KB of D-cache, just bits [31:10] are valid, bit 9 is ignored.

In the RC32334, the *W* (write-back) bit—not the cache state—indicates whether or not the primary cache contains modified data that must be written back to memory.

Note: There is no hardware support for cache coherency. The only cache states used are Dirty Exclusive and Invalid.

Notes

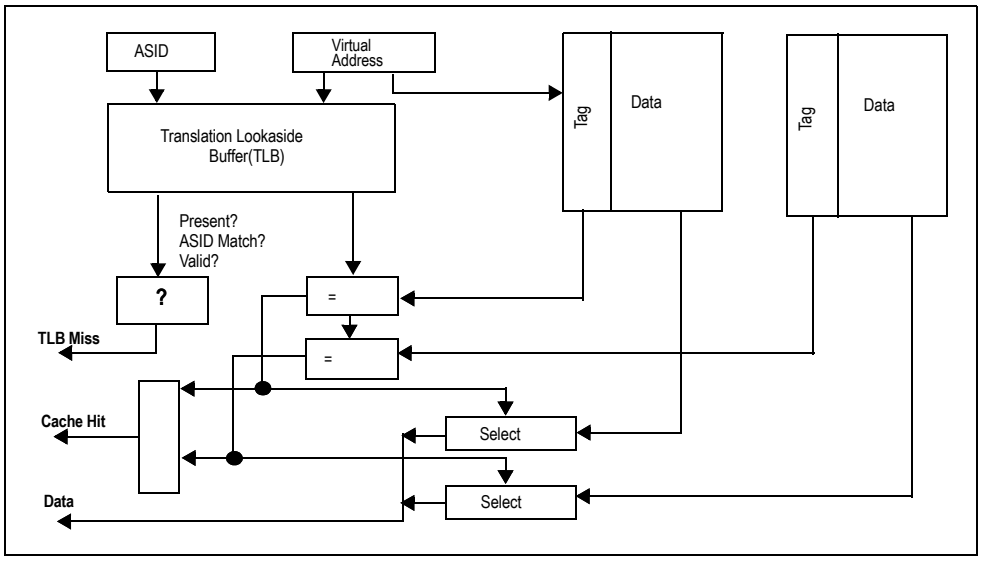


Figure 7.4 Conceptual Primary Cache Lookup Sequence

Accessing the Primary Caches

Figure 7.5 shows the virtual address (VA) index into the primary caches. For the RC32334 the instruction cache is 8kB and the data cache is 2kB.

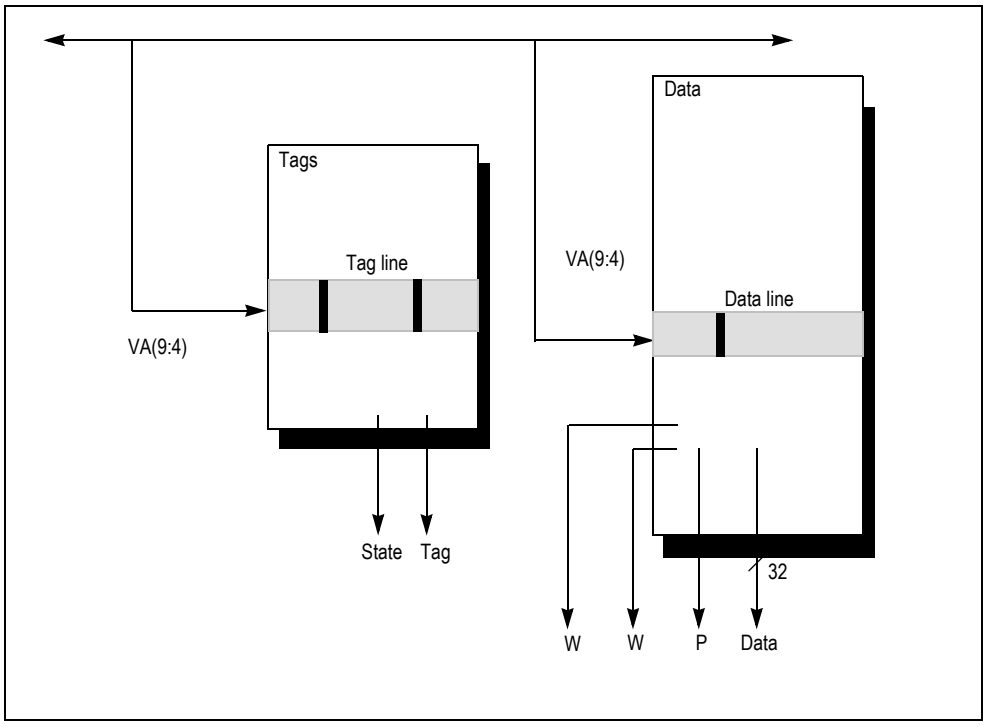


Figure 7.5 Primary Cache Data and Tag Organization

Notes

Primary Cache States

The terms below are used to describe the *state* of a cache line[1]:

- ***Exclusive****: a cache line that is present in exactly one cache in the system is exclusive. This is always the case for the RC32334. All cache lines are in an exclusive state.*
- ***Dirty****: a cache line that contains data that has changed since it was loaded from memory is dirty.*
- ***Clean****: a cache line that contains data that has not changed since it was loaded from memory is clean.*
- ***Shared****: a cache line that is present in more than one cache in the system. The RC32334 does not provide for hardware cache coherency. This state will not occur during normal operations.*

The RC32334 supports the four cache states shown in Table 7.4. Under normal operations, the only states that will occur in the RC32334, are the Dirty Exclusive and Invalid states.

Note: Although valid data is in the Dirty Exclusive state, it may still be consistent with memory. One must look at the dirty bit, W, to determine if the cache line is to be written back to memory when it is replaced.

Each primary cache line in the RC32334 system is in one of the states described in Table 7.4.

Cache Line State	Description
Invalid	A cache line that does not contain valid information must be marked invalid, and cannot be used. A cache line in any other state than invalid is assumed to contain valid information.
Shared	A cache line that is present in more than one cache in the system is shared. This state will not occur for normal operations.
Clean Exclusive	A clean exclusive cache line contains valid information and this cache line is not present in any other cache. The cache line is consistent with memory and is not owned by the processor (see “Cache Line Ownership” on page 7-6 in this chapter). This state will not occur for normal operations.
Dirty Exclusive	A dirty exclusive cache line contains valid information and is not present in any other cache. The cache line may or may not be consistent with memory and is owned by the processor (see “Cache Line Ownership” on page 7-6 in this chapter). Use the W bit to determine if the line must be written back on replacement.

Table 7.4 Primary Cache States

Primary Cache States

Each primary data cache line is normally in one of the following states:

- *invalid*
- *dirty exclusive.*

Each primary instruction cache line is in one of the following states:

- *invalid*
- *valid.*

Cache Line Ownership

The processor is the owner of a cache line when it is in the dirty exclusive state, and is responsible for the contents of that line. There can only be one owner for each cache line.

The ownership of a cache line is set and maintained through the rules described below.

- *A processor assumes ownership of the cache line if the state of the primary cache line is dirty exclusive.*

1. A cache line is the smallest unit of information that can be fetched from memory to be filled into the cache. A primary cache line is 16 bytes (4 words) in length and is represented by a single tag. Upon a cache miss in the primary cache, the missing cache line is loaded from main memory into the primary cache.

Notes

- *A processor that owns a cache line is responsible for writing the cache line back to memory if the line is replaced during the execution of a Write-back or Write-back Invalidate cache instruction if the line is in a write-back page. The Cache instruction is explained in Appendix A.*
- *Memory always owns clean cache lines*
- *The processor gives up ownership of a cache line when the state of the cache line changes to invalid.*

Therefore, based on these rules and that any valid data cache line is in the Dirty Exclusive state (under normal operating conditions), the processor is considered to be the owner of the cache line.

Cache Write Policy

The RC32334 caches use the same write algorithms defined for the RC4700. These algorithms are specified by the "C" bits[1] of a TLB entry or through the K0 field of the status register.

The RC32334 processor manages its primary data cache by using either a write-back or a write-through policy selected on a per-page basis through the TLB. In a write-back cache, the data is not written back to memory until the cache line is replaced.

A write-through policy means the store data is written to the cache and to memory. Due to the write buffer, the write of the data to memory may not occur at the same time as the write to cache.

For a write-back entry, if the cache line is valid and has been modified (the *W* bit is set), the processor writes this cache line back to memory when the line is replaced, either in the course of satisfying a cache miss or during the execution of a Write-back or Write-back Invalidate CACHE instruction.

For a write-through entry, whenever a store hits in the cache line, the data is also written to memory via the write buffer. The store will not set or clear the *W* bit for a write-through cache line. This is to allow a different virtual address that maps to the same physical address and with a write-back policy to still set the *W* bit.

For a miss to a write-through line, the action taken will be determined by the write-allocation policy. For a write-allocate entry, the cache line is first retrieved from memory and the store will then continue. A no write-allocate entry will just post the write to the system interface, via the write buffer, in the same manner as an uncached write.

Store Buffer

To implement the write-back cache, the store instructions to cacheable memory operation must include a read/write sequence to the cache; the read first determines whether the line is cache resident; the subsequent write updates the appropriate bytes, dirty bit, and parity bits.

To allow back-to-back data cache access, the RC32334 implements the same store buffer concept included with IDT's 64-bit RISController. This avoids extra stalls after store instructions to complete the read-modify-write sequence required to update the cache line.

Cache Replacement Policy

The RC32334 uses the following algorithm to select a cache line from the available sets for replacement:

- *If both lines are invalid, select set A.*
- *If only one set is marked invalid, select that set.*
- *If one set is locked, select the other set.*
- *If both sets are locked, select set A*[2]*.*
- *If both sets are valid and unlocked, select the line which has been in the cache the longest. Each cache line contains a "FIFO" bit to help determine which line was least recently replaced.*

1. See Table 5.1 in Chapter 5 of this manual for bit values and attribute assignment.
2. This is an erroneous condition; however, the RC32334 handles this case deterministically.

Notes

Cache Initialization

The RC32334 includes 2kB of 2-way set associative data cache that corresponds to an address range between 0x000 and 0x7fc. The cache index offset for set A is 0x000, while the cache index offset for set B is 0x1000. To avoid any cache initialization problems, please select one of the following two initialization methods:

1. Initialize index location 0x000-0x3fc for set A and then 0x1000-0x13fc for set B.

or

2. Initialize as if the data cache were at least 8k large.

The I-cache tag should also be initialized using "cache op" instruction with the index location 0x0000-0x0FFC for set A, and then 0x1000-0x1FFC for set B.

Cache Locking

The RC32334 also supports a cache-locking feature that can be used to lock critical sections of code and/or data into on-chip caches to guarantee quick access.

A portion of a cache is said to be *locked* when a particular piece of code or data is loaded into a cache location that will not be selected later for refill by other data. The locking feature of the RC32334 is on a per-line basis; that is, the kernel may set status register control bits that allow individual cache lines to be locked in the cache.

Locked cache lines can be changed by any of the following operations or conditions:

- *cache operations*
- *store operations to cached virtual address*
- *if they become valid.*

When to use Cache Locking

Cache locking is useful in the following cases:

- *a portion of code must reside in cache permanently (for example, time-critical exception vectors) for real-time performance*
- *a given section of code is executed frequently and can fit inside a portion of the instruction cache*
- *a given section of data is accessed frequently and can fit inside the data cache (for example, tables containing routing information in an embedded network application).*

In the RC32334, both the Instruction and Data cache are two-way set associative, with set A and set B. By setting the DL or IL bit in the Status register of CP0, a refilled cache line of a selected set, at that time, can be locked in the appropriate cache; therefore, a future fill into this cache line will always use the other set. Furthermore, if one set of a cache line has already been locked, the second attempt to lock this cache line will be ignored.

As previously noted, a Data store operation to locked data will update the D-cache contents; locking merely prevents the cache line contents from being replaced by the contents of a different physical location. The locked cache line can be unlocked by using a Cache operation to invalidate that line. Anytime the *valid* bit of a cache line is cleared, the *lock* bit is cleared simultaneously. The basic algorithm presented here consists of the following three steps.

1. Set the appropriate cache-lock enable bit(s).
2. Load the critical code/data into the cache(s).
3. Clear the appropriate cache lock enable bit(s).

Example: Data Cache Locking

For this example, assume an application in which a table must be kept in cache. After completing the initialization of data structures, etc., in the start-up code, the DL bit in the Status register can be set to enable the cache line locking, perform reads through cached addresses to load the data into the data cache, and then—to prevent further cache locking— clear the DL bit. A sample code fragment for the Data Cache Locking operation follows:

Notes

```
        .set noreorder
    jal         flush_cache          /* Flush the cache */
    mfc0    a0, C0_SR          /* Get old SR value */
    li      a1, SR_SET_DL      /* SR_SET_DL = 0x00100000 */
    or      a0, a0, a1
    mtc0    a0, CO_SR      /* Set the Lock bit for data cache */
    nop
    nop
    nop                 /* 3 nops: safety against CP0 hazard */

    la   t0, critical_table   /* This table should always be in      cache */
         li     t1, table_size     /* Size of table in bytes */
         li     t2, 0          /* Number of bytes read into cache */

    1:    lw    a0, 0(t0)
    addiu  t2, 4
    bneq   t2, t1, 1b         /* Loop back till done */
    addiu  t0, 4              /* bump read address */

    mfc0   a0, C0_SR          /* Get old SR value */
         li     a1, SR_CLR_DL       /* SR_CLR_DL = 0xffefffff */
         and    a0, a0, a1
    mtc0   a0, C0_SR   /* Clear the Lock bit for data cache */
    nop
    nop
         nop                /* 3 nops: safety against CP0 hazard */
```

Example: Instruction Cache Locking

For this example, assume an application in which a critical function must be kept in cache. Also assume that the size of the function is known. (If not known, the size can be determined by generating a disassembly of the object file.)

After completing the initialization of data structures, etc., in the start-up code, the IL bit of the Status register can be set to enable cache line locking, perform the FILL operation in the CACHE instruction that will fill the instruction cache with the critical function, and then—to prevent further cache locking—clear the IL bit.

A sample code fragment for the Instruction Cache Locking operation follows:

```
        .set noreorder
    jal                                         flush_cache         /* Flush the cache */
         la         t0, 1f          /* Get address of label '1' */
         li         t1, 0xA0000000
         or         t0, t0, t1
         jr     t0        /* Uncached execution from now onwards */
         nop

    1:  la t0, func_start_addr    /* Start address of critical code */
```

Notes

```
        li      t1, func_size       /* Critical code size */
        li      t2, 0           /* Number of words read into cache */
        mfc0    a0, C0_SR           /* Get old SR value */
        li      a1, SR_SET_IL       /* SR_SET_IL = 0x00080000 */
        or      a0, a0, a1
mtc0  a0, C0_SR     /* Set Lock bit for instruction cache */
        nop
        nop
        nop

2:                                  cache   Fill_I, 0(t0)       /* Fill Operation */
        addiu   t2, 4
        bneq    t2, t1, 2b          /* Loop back till done */
        addiu   t0, 4               /* bump read address */

        mfc0    a0, C0_SR           /* Get old SR value */
        li      a1, SR_CLR_IL       /* SR_CLR_IL = 0xfff7ffff */
        and     a0, a0, a1
        mtc0  a0, C0_SR   /* Clear Lock bit for instruction cache */
        nop
        nop
        nop
        nop
        nop             /* 5 nops: safety against CP0 hazard */
la      v0, 3f
jr      v0
nop

3:              /* Resume execution in mode as linked */
```

RC32334 Internal Bus

Notes

Introduction

The RC32334 is an integrated processor which is logically an integration of two discrete existing IDT devices; the RC32364 standalone CPU and the RC32134 system controller. The bus that connects the two discrete devices together, is replaced internally in the RC32334 integrated processor, referred to in this manual as the RC32300 CPU bus.

Although this bus in not visible external to the RC32334, there are certain registers that need to be configured by the user. This chapter discusses these aspects.

Generally, the RC32300 CPU bus features a multiplexed address/data bus and a number of associated control signals. A device controller module latches and decodes the address information originating from the RC32300 CPU core to determine which memory, I/O or peripheral module is being accessed, per the internal address map of the RC32334.

List of Features for RC32300 CPU Bus

- *Internal 32-bit physical addressing*
- *Internal 32-bit data bus*
- *Internal command/data protocol*
- *Internal generic read/write and burst read/write protocols*
- *Internal bus arbitration modes*
- *Internal chip select generation*

Block Diagram

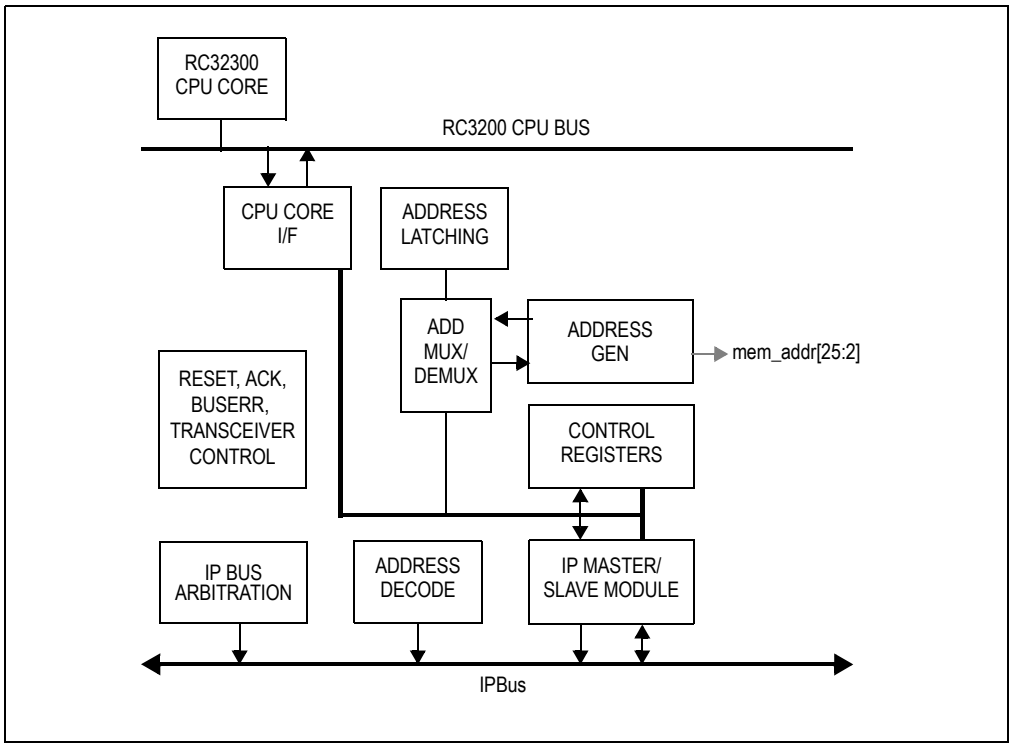


Figure 8.1 IP Bus Bridge Block Diagram

Notes

Functional Overview

The RC32334's internal bus bridge provides a translation from the RC32300 CPU BIU interface control/address bus to the internal IP bus. The RC32334's internal IP bus is a synchronous command oriented bus that connects multiple DMA masters and all of the peripheral slaves. The bridge also provides address decoding and generates all address lines to the external memory and I/O peripherals. The internal bus bridge contains the following three main components:

- *Address module*
- *Data module*
- *Control module.*

Address Module

The Address module of the internal bus bridge translates either the RC32300 CPU core generated addresses or addresses generated by internal modules (DMA Controller, PCI Interface) to the primary address lines, for use by the external Memory system and I/O peripherals. The Address module also generates decoding for all external chip and internal module chip selects.

Address Incrementer

The system address that is used to interface to external memory such as PROM or DRAM is incremented after each word (32-bit) access. RC32300 CPU core initiated reads must be incremented with subblock ordering, which is shown in Figure 8.2. Other accesses—including those generated from the DMA Controller and PCI bridge interface as well as all writes—increment addresses with linear ordering.

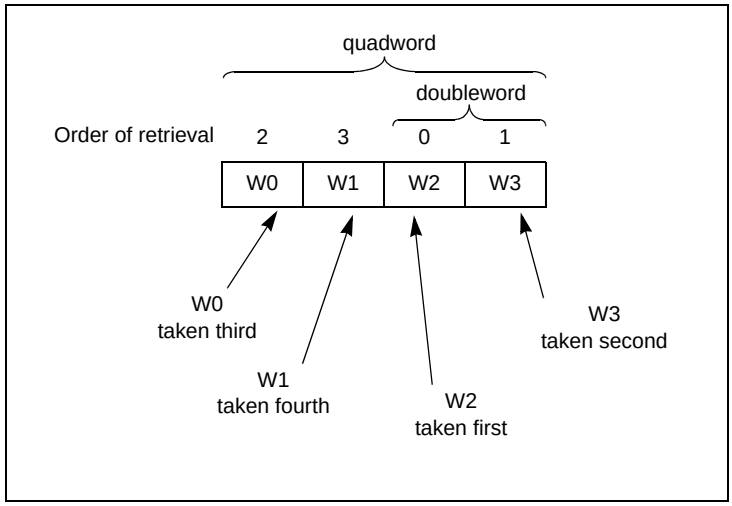


Figure 8.2 Subblock Ordered Data Retrieval

The RC32334 makes a system simplification and assumes that all accesses are 32-bit. Thus, both the 8- and 16-bit PROMs must connect Addr[3:2], BE_n[1], and BE_n[0] directly to the RC32334 if mem_addr[3.2] are used, in accordance with the table within Table 1.2, Pin Descriptions. The RC32334 will inadvertently increment (in subblock order) mem_addr[3:2] on each CPU byte access. For more information, refer to the Using 8- or 16-bit Boot PROMs section of Chapter 10 for more information.

Address MUX

The RC32334 uses the first set of address lines (mem_addr[25:2]) to provide the address to external memory (ROM, EPROM, SRAM) and I/O peripherals. SDRAM, the address lines (mem_addr[15:2]) provide the DRAM row and DRAM column addresses. An address mux is implemented to map the address of the transaction on the proper address lines. More details on the address mapping convention for these operations are provided in the Memory Controller section in Chapter 10 and the Synchronous DRAM Controller section in Chapter 11.

Notes

Address Decode

Address Decode takes the internally latched address and generates both external chip selects and internal module chip selects, based on the memory map. Some of the external chip selects are fixed to a particular physical address range. The address range of the other external chip selects is software programmable via the Memory Base Address Registers, Memory Base Mask Registers, DRAM Base Address Registers, and DRAM Base Mask Registers. These registers are described in the Memory-IO and DRAM chapters of this manual. Also, the Reset initialization requires that the Address Latch Timing Register be setup to optimize timing by choosing a 1 clock or 2 clock address decode, as described later in this chapter.

During the first clock of a transaction, the RC32334 decodes the address and compares it to the base address registers/constants from SDRAM, MEMORY, PCI, other peripherals, and controller register banks. Furthermore, on transactions selecting memory spaces within the SDRAM and MEMORY controllers, each compare bit is masked to determine whether that bit is involved in the compare. The mask operation allows multiple banks of memory to be in a linear contiguous address space.

Note: In Figure 8.3 through 8.5, cpu_ale_n and cpu_ad[31:0] signals are internal to the RC32334 and are shown for reference only. They are generated by the RC32300 CPU core.

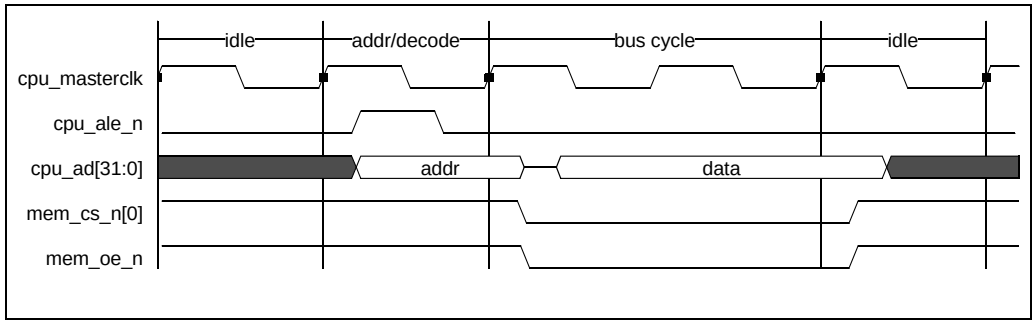


Figure 8.3 Address Latch Time with Fast Decode Setting

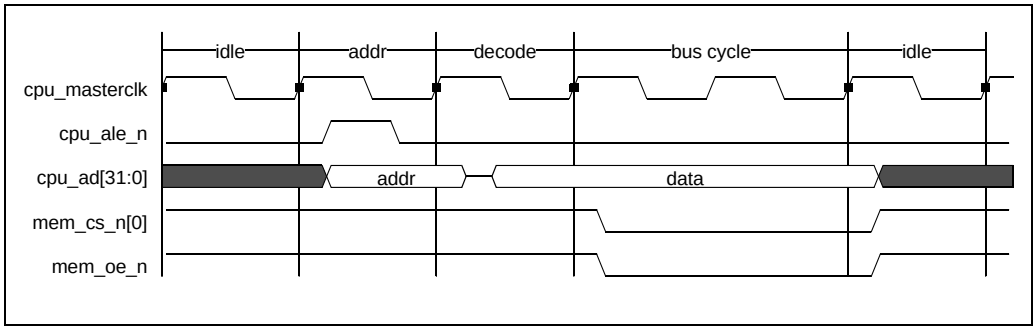


Figure 8.4 Address Latch Time with Slow Decode Setting

Data Module

During read or write transactions to external memory and peripherals generated by the internal RC32300 CPU core, the RC32334 bus generates all control and address signals. Thus, the RC32334 provides sufficient control signals to enable data driven from memory.

CPU Read/Write Operations

During the data phase of a write operation, the RC32334 is the master of the bus and drives the data on cpu_ad[31:0] bus (see timing in Figure 8.5). However, during the data phase of a read operation, external memory or I/O becomes bus master and drives the data on the mem_data[31:0] which is latched internally inside the RC32334 onto the cpu_ad[31:0] bus.

Notes

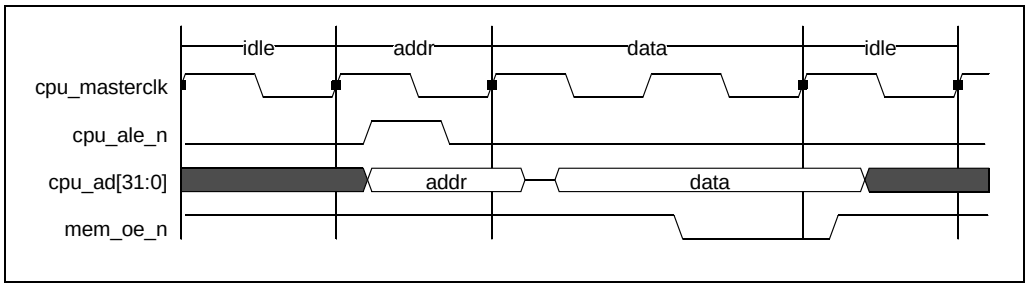


Figure 8.5 RC32334 cpu_ad[31:0] Data Phase

DMA Read/Write Operations

During DMA operations, or when the PCI bus is accessing main memory, the RC32300 CPU core is not involved in the transaction and the RC32334 is master of the internal cpu_ad[31:0] bus. During DMA write operations, the RC32334 drives the address and the control signal to the memory and I/O peripherals and drives the data on the mem_data[31:0] bus. During DMA read operations, the RC32334 drives the address and control signals to the memory or I/O peripherals, which in turn returns the data to the RC32334 by driving the mem_data[31:0] bus.

Arbitration

The RC32300 CPU core is the default bus master on the internal cpu_ad[31:0] bus. A RC32334 on-chip peripheral module requests the bus from the CPU core when DMA operations occur, or when the PCI bridge reads or writes to the main memory. The RC32334 implements an internal arbiter to arbitrate for the cpu_ad[31:0] bus in the following two options:

1. Fixed (CPU, (0,1,2,3,PCI))
2. Round robin after each grant: (CPU,(0,1,2,3,PCI)), (CPU,(1,2,3,PCI,0)), (CPU,(2,3,PCI,0,1)),...

The cpu_ad[31:0] bus protocol enables the CPU core to request the bus after it has been granted by de-asserting the busgnt_n signal (an internal signal not visible to devices and external to the RC32334). Thus, between any two DMA accesses, the CPU will typically issue an access. However, if a DMA request is pending, the CPU will always give the bus up after its present transaction completes.

Memory Port Sizing

The RC32334 divides the physical memory space into 12 regions. The port width of each region can be configured in the RC32334's Port-Width Control register. Physical memory space in the RC32334 is divided into 12 distinctive regions (typical mapping of the physical regions is shown in Table 1.4).

The port-width sizes of the DRAM (A,B,C,D) and PCI (J & K) regions are always 32-bit; however, the port-width sizes of the memory and I/O regions (H & I) are programmable through the Port-Width field of the Memory Controller register, as described later in this chapter.

Bus Turnaround (BTA) Register

The RC32364 BIU core includes a BTA register that specifies, per memory region, the number of clock cycles the CPU core will wait before issuing a new transaction after completing a read operation. A similar BTA feature is included in the RC32334 peripheral controller.

This BTA feature allows slow-to-disable-data devices such as EPROM to share a data bus with other devices. RC32334 does not allow a transaction to follow a read in less than the BTA value setup per address block. Similar to the RC32300 CPU core BTA Register, all RC32334 BTA settings in the peripheral controller are set to their maximum of '3' at reset. The software operating system kernel should program this register immediately as part of the power-up/reset boot sequence. A field description table (Table 8.8) and format diagram (Figure 8.8) for this register are provided in the Register Descriptions section of this chapter.

Notes

Watchdog Timer

As part of the Timer module, a Watchdog timer is included that works as a safeguard mechanism to assist in detecting runaway software. The Watchdog timer will issue an internal cpu_reset_n (warm reset) to the CPU core if the Watchdog Timer rolls over. When the RC32334 issues a warm reset, it does not affect the RC32334 reset boot-mode settings.

Normally, the software operating system kernel must occasionally reset the Timer Count Register (to `0x0000_0000`) so that the rollover does not occur. In Standard-boot mode, the WatchDog Timer function is enabled at reset. If the OS kernel chooses not to implement this function, the boot code must, at a minimum, disable this function by writing to the BusError Control Register of the IP Bridge. A WatchDog Timer Status bit indicates if the last reset was caused by the WatchDog Timer. In PCI-boot mode, the WatchDog Timer is disabled at reset.

Bus Time-Out Counters

Two 16-bit software programmable bus time-out counters are also provided, each with its own comparator: software programmable abort, including externally generated wait-states. There is a software programmable enable/disable bit. In addition to the abort, an internal interrupt is generated. A bus time-out terminates the present data of a memory transaction, causing buserror_n and ack_n (both signals internal to the RC32334 interconnecting the System Controller to the CPU) to be returned. The bus time-out setting is typically calculated from the maximum burst length of the RC32334 at 4 words using its slowest transaction. Typically, an 8-bit boot EPROM burst transfer of 4 words (16 bytes) is the longest possible transfer. When a bus time-out occurs, the present physical address is latched into the Bus Error Register.

Bus Error Timers

The RC32334 includes two bus error timers. The first is used for RC32364 BIU core bus time-outs and will time-out if the CPU bus is held too long. Because the present implementation of RC32334 always gets the CPU bus when an IP access is in process, a CPU bus time-out also causes an internal IP bus time-out.

For systems that must distinguish between CPU and IP accesses, an optional IP bus time-out timer is provided. This timer will only assert if the IP bus is held too long, regardless of the CPU bus time-out. Most systems will not need to use this timer and can reassign the timer for general purpose usage. If used, the IP bus time-out timer is typically set to 1 more than the CPU bus time-out, so that the two cases can be distinguished by the bus error status bits.

Register Descriptions

Address	Register
FFFF_E200	CPU Port Width Register
FFFF_E204	CPU BTA Register
FFFF_E208	CPU BusError Address Register

Table 8.1 CPU Bus Interface Control Registers

Address	Register
1800_0000	BTA Register
1800_0004	Address Latch Timing Register
1800_0008	Arbitration Register

Table 8.2 CPU to IP Register Addresses and Descriptions (Part 1 of 2)

Notes

Address	Register
1800_0010	BusError Control Register
1800_0014	BusError Address Register
1800_0018	SysID Register

Table 8.2 CPU to IP Register Addresses and Descriptions (Part 2 of 2)

Interface Control Registers

The following three interface control registers are used in the RC32334:

- *The CPU Port-Width Control register controls attributes of the various memory systems and is used to interface the RC32334 to varying width memory regions.*
- *The CPU Bus Turnaround (BTA) control register controls the bus turnaround cycle(s) for the various memory systems. The RC32334 divides the physical address space into various sub-regions, each of which features independently programmable bus turnaround cycle(s).*
- *The CPU Bus Error Address Register holds the physical address of the transfer that signalled the most recent bus error.*

CPU Port-Width Control Register: Virtual Address 0xFFFF_E200

The RC32334 divides the physical address space into various sub-regions, each of which features independently programmable port widths. At reset, all memory widths are set to the width of the boot prom. Software may then re-program the widths of various regions to meet the actual system implementation.

Using the Port Width Control register allows software to be fully independent of the actual system implementation; software may then treat all references as if the memory was 32-bits wide and relies on the RC32334 to manage the bus interaction with the actual memory system to satisfy this model.

The format of the CPU Port Width Control register is shown in Figure 8.6. Table 8.3 lists the register's fields and content descriptions.

Note: Region G should always be programmed to 32-bit port width during boot code initialization before the System Controller Registers are used.

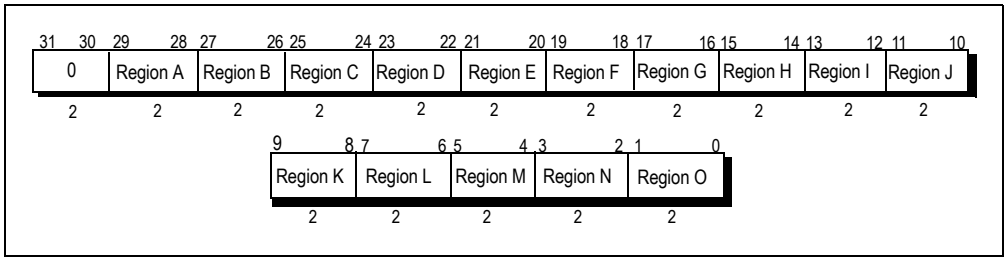


Figure 8.6 Format of CPU Port Width Control Register

Field	Description
0	Reserved
RegionA	Width of region RegionA
RegionB	Width of region RegionB
RegionC	Width of region RegionC
RegionD	Width of region RegionD
RegionE	Width of region RegionE

Table 8.3 Port Width Control Register Field Definition (Part 1 of 2)

Notes

Field	Description
RegionF	Width of region RegionF
RegionG	Width of region RegionG
RegionH	Width of region RegionH
RegionI	Width of region RegionI
RegionJ	Width of region RegionJ
RegionK	Width of region RegionK
RegionL	Width of region RegionL
RegionM	Width of region RegionM
RegionN	Width of region RegionN
RegionO	Width of region RegionO

Table 8.3 Port Width Control Register Field Definition (Part 2 of 2)

Width fields are encoded as shown in Table 8.4.

Width(1) (MSB)	Width(0) (LSB)	Port Width
0	0	8 bits
0	1	16 bits
1	0	32 bits
1	1	Reserved

Table 8.4 Encoding of 8-, 16-, and 32-bit Port Widths

The address ranges served by each named region are listed in Table 8.5. The memory space is divided as follows:

- *The 512MB of kseg0/1 are divided into eight 64MB sub-regions, each of which can have independent widths. Thus, four widths can be reached cacheably, and four widths can be reached uncacheably. The cache management algorithm for kseg0 is specified in the k0 field of the status register.*
- *The remaining memory space—3.5GB—is divided into seven equal sections of 512MB, each of which can have independent widths. In addition, the cache attributes of each page within the region can be controlled using the TLB.*

Region Name	Physical Address (31:26)	Comments
RegionA	0000 00	64MB
RegionB	0000 01	64MB
RegionC	0000 10	64MB
RegionD	0000 11	64MB
RegionE	0001 00	64MB
RegionF	0001 01	64MB
RegionG	0001 10	64MB
RegionH	0001 11	64MB
RegionI	001x xx	512MB
RegionJ	010x xx	512MB
RegionK	011x xx	512MB

Table 8.5 Memory Region Address Ranges (Part 1 of 2)

Notes

Region Name	Physical Address (31:26)	Comments
RegionL	100x xx	512MB
RegionM	101x xx	512MB
RegionN	110x xx	512MB
RegionO	111x xx	512MB

Table 8.5 Memory Region Address Ranges (Part 2 of 2)

CPU Bus Turnaround (BTA) Control Register: Virtual Address 0xFFFF_E204

At reset, all memory sub-regions will be programmed to the maximum of 3 turnaround cycles, and software may then re-program this register to achieve maximum system performance.

The format of the BTA register is shown in Figure 8.7. This register's fields and content descriptions are listed in Table 8.6.

Note: Region G should always be programmed to a BTA=1 during boot code initialization. In general, most Regions can use BTA=1. Note that the T recovery cycle, shown in Figure 8.9, is used by the RC32334 to pre-charge the mem_data bus during CPU-generated accesses. Thus, the BTA=0 setting should never be used unless proper bus isolation techniques are utilized, such as with Q-logic transceivers.

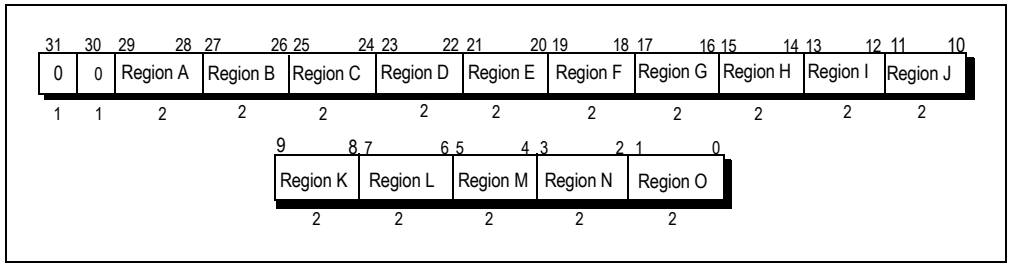


Figure 8.7 CPU Bus Turnaround (BTA) Control Register Format

Field	Definition
0	Reserved
0	Reserved
RegionA	Turnaround cycle(s) of region RegionA
RegionB	Turnaround cycle(s) of region RegionB
RegionC	Turnaround cycle(s) of region RegionC
RegionD	Turnaround cycle(s) of region RegionD
RegionE	Turnaround cycle(s) of region RegionE
RegionF	Turnaround cycle(s) of region RegionF
RegionG	Turnaround cycle(s) of region RegionG
RegionH	Turnaround cycle(s) of region RegionH
RegionI	Turnaround cycle(s) of region RegionI
RegionJ	Turnaround cycle(s) of region RegionJ
RegionK	Turnaround cycle(s) of region RegionK

Table 8.6 CPU Bus Turnaround (BTA) Control Register Field Descriptions (Part 1 of 2)

Notes

Field	Definition
RegionL	Turnaround cycle(s) of region RegionL
RegionM	Turnaround cycle(s) of region RegionM
RegionN	Turnaround cycle(s) of region RegionN
RegionO	Turnaround cycle(s) of region RegionO

Table 8.6 CPU Bus Turnaround (BTA) Control Register Field Descriptions (Part 2 of 2)

The turnaround cycle(s) is encoded as shown in Table 8.7. After a read access, it indicates the minimum number of Turnaround clock cycles that must occur before the next read or write access can occur. Figure 8.9 shows the timing of the BTA cycle. Note the clock cycle denoted by the T_{TA} symbol indicates the minimum number of Turnaround cycles that must occur after a read access.

TA(1) (MSB)	TA(0) (LSB)	Turnaround cycle(s)
0	0	0 cycle
0	1	1 cycle
1	0	2 cycles
1	1	3 cycles

Table 8.7 Width Encoding of Bus Turnaround Cycles

CPU Bus Error Address Register (Read Only): Virtual Address 0xFFFF_E208

This is a read only register that holds the address that caused the bus error. Any attempts to write to this register will not change its value, which is not defined before the Bus Error is sampled.

BTA Control Register

Note: Although this register exists, it is not functional. Refer to the RC32334/RC32332 Device Errata, located at www.idt.com, for additional information.

Bus turnaround time refers to that period of time after a read transaction ends before the next transaction can begin. This time period allows the memory, or its transceiver just read, to tri-state its data from the AD bus before the next address is driven out by the CPU as shown in Figure 8.9 on page 8-11.

On the RC32334, the BTA register is used within DMA transactions, whenever a read occurs, and does not allow a transaction to follow a read in less than the BTA value setup per address block. At reset, all memory subregions are programmed to the maximum of 3 turnaround cycles, and software should then reprogram this register to achieve maximum system performance. A setting of '1' is typical.

After a DMA descriptor burst read, no Bus Turnaround (BTA) clocks are inserted and a CPU address may appear on the mem_data [] bus as soon as 2 clocks after the DMA descriptor read.

Note: Region G, which sets the BTA for the RC32334 internal register space, must be programmed to a setting of '1' or greater.

The format of the BTA control register is shown in Figure 8.8. This register's fields and content descriptions are listed in Figure 8.8. The regions shown correspond to the BTA regions described in the RC32334's BTA register.

Notes

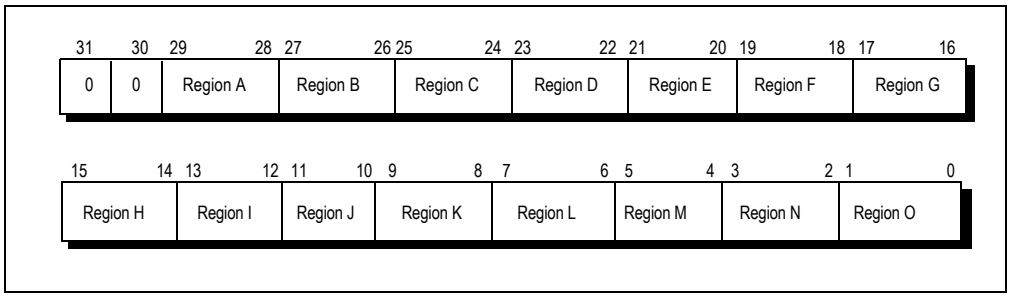


Figure 8.8 Bus Turnaround (BTA) Control Register Format

Field	Definition
0	Reserved
0	Reserved
Region A	Turnaround cycle(s) of region Region A
Region B	Turnaround cycle(s) of region Region B
Region C	Turnaround cycle(s) of region Region C
Region D	Turnaround cycle(s) of region Region D
Region E	Turnaround cycle(s) of region Region E
Region F	Turnaround cycle(s) of region Region F
Region G	Turnaround cycle(s) of region Region G[1]
Region H	Turnaround cycle(s) of region Region H
Region I	Turnaround cycle(s) of region Region I
Region J	Turnaround cycle(s) of region Region J
Region K	Turnaround cycle(s) of region Region K
Region L	Turnaround cycle(s) of region Region L
Region M	Turnaround cycle(s) of region Region M
Region N	Turnaround cycle(s) of region Region N
Region O	Turnaround cycle(s) of region Region O

Table 8.8 Bus Turnaround (BTA) Control Register Field Descriptions

1. It is mandatory to program region G for at least 1 cycle turnaround.

The turnaround cycle(s) is encoded as shown in Table 8.9. Figure 8.9 shows the timing of the BTA cycle.

TA(1) (MSB)	TA(0) (LSB)	Turnaround cycle(s)
0	0	0 cycle
0	1	1 cycle
1	0	2 cycles
1	1	3 cycles (default)

Table 8.9 Width Encoding of Bus Turnaround Cycles

Notes

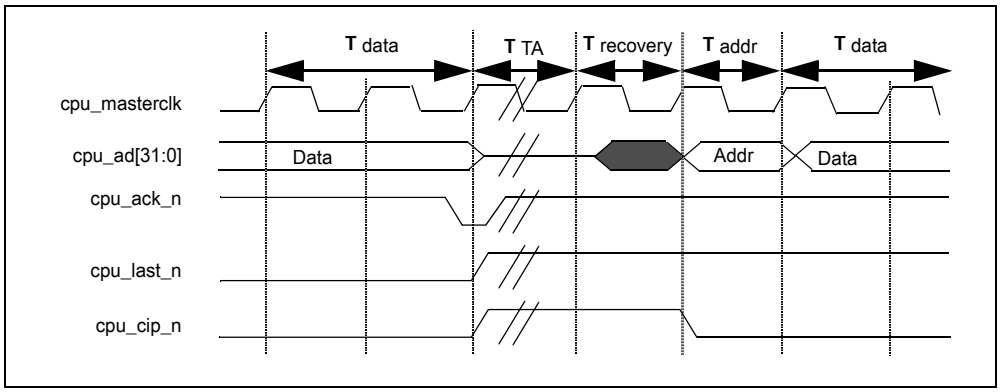


Figure 8.9 Timing of Bus Turnaround Cycle(s) (Example of 1 Cycle BTA)

These are internal signals and are shown here for reference purposes only.

Address Latch Timing Register

The Address Latch Timing Register delays initial address decode from the RC32300 CPU core BIU by 1 clock until the first rising clock edge after the internal cpu_ale asserts. This mode pipelines the address decode such that 50MHz and faster systems have ample setup time to properly select a register/memory before a synchronous clock edge. At reset, the default is to take the delay. Thus systems that are running at less than 50MHz must reprogram the Memory Controller Address Latch Timing bit to “decode address” so that performance is increased. The address latch times with fast and slow decode settings are shown in Figures 8.3 and 8.4.

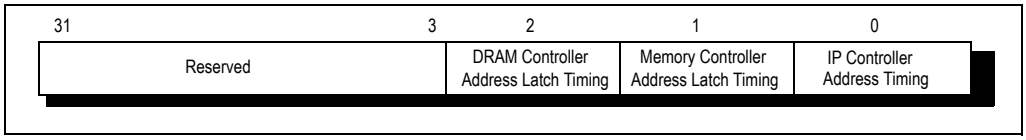


Figure 8.10 Address Latch Timing Register

Bit	Field Name	Description
31:3	Reserved	
2	DRAM Controller Address Latch Timing	Setting: 1 — Delay address decode by 1 clock for systems running at > 67 MHz; Setting: 0 — Don’t delay address decode by 1 clock, for systems running at ≤ 67 MHz (default)
1	Memory Controller Address Latch Timing	Setting: 1 — Delay address decode by 1 clock, for systems running at ≥ 50 MHz (default); Setting: 0 — Decode address on falling edge of ALE, for systems running at < 50 MHz
0	IP Register Controller Address Latch Timing	Reserved to 1 to delay the address decode by 1 clock.

Table 8.10 Address Latch Timing Bit Field Descriptions

Notes

Arbitration Register

The Arbitration register is used to select the arbitration method used for prioritizing access to the CPU bus by the CPU core, the PCI bridge and the DMA controller channels. For the specific details of this operation, refer to the DMA Controllers section in Chapter 13.

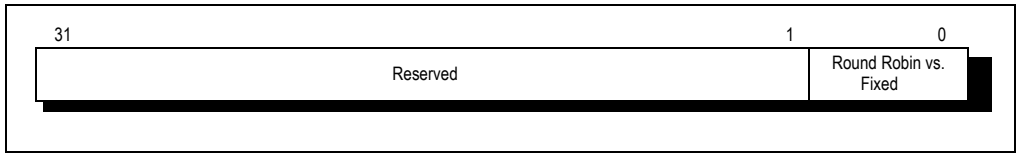


Figure 8.11 Arbitration Register Field

Value	Description
1	Round Robin arbitration
0	Fixed Priority arbitration (default)

Table 8.11 Arbitration Field Values and Action Description

BusError Control Register

The Bus Error register stores the current address of any transaction—Read, Write, CPU generated, DMA generated, or PCI generated. Bus errors occur if a bus time-out occurs and no memory space is selected. For CPU generated transactions, the RC32334 will assert the buserr_n to the CPU and will terminate the transaction. The fields of the BusError register are shown in Figure 8.12. The function of each field is listed in Table 8.12.

Note: If the PCI-boot mode is selected at reset time, the CPU BusError, IP BusError, and Watchdog bits are disabled.

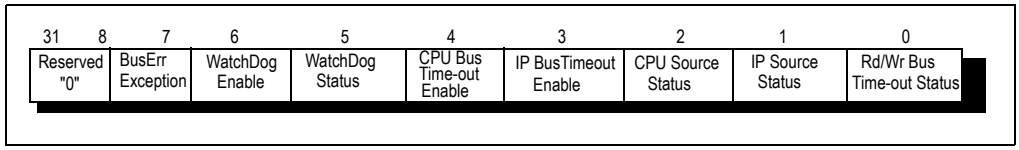


Figure 8.12 BusError Control Register Fields

BusError Address Register

RC32334's BusError Address Register (see Figure 8.13) is similar to the RC32334 on-chip CPU Bus Error Address Register, and for CPU generated transactions, the value should be the same in both registers. The RC32334 Bus Error Register is also used on DMA operations and bus time-out errors. The Interrupt Pending Register is used to first determine whether an error occurred as a result of a bus error (non-decoded address) or a bus time-out (acknowledge never returned). The default value of this register is 0x0000_0000.

Note: On bus errors, if the CPU transaction was a read, a bus exception is generated in addition to an interrupt. If the CPU transaction was a write, an interrupt is generated but no bus exception is taken. This behavior occurs because the CPU write buffer cannot re-align the original store instruction issuance with the bus error.

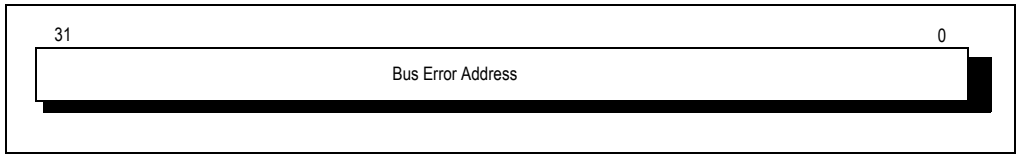


Figure 8.13 BusError Address Register

Notes

Bits	Field Name	Description
31:8	Reserved to "0"	For future compatibility, must be written as "0".
7	BusError Read Exception Disable	On a BusError, if this bit is low, a physical bus error and an interrupt (if enabled) to the CPU is generated on CPU accesses, thus terminating the CPU access. Also, on a physical bus error, CPU reads take an exception/interrupt, while CPU writes take an interrupt. If this bit is high, then neither a bus error nor a bus error read exception is generated, the access is terminated, and an interrupt (if enabled) is generated. **Value** / **Description** 1 – Disabled 0 – Enabled (default)
6	WatchDog Enable	When WatchDog Enable is enabled, when the WatchDog Timer reaches its compare count and overflows, a warm reset will be generated to the CPU core. To prevent the WatchDog Timer from generating a reset, RC32334 systems must have enough OS kernel support to occasionally zero out the WatchDog Timer Count Register or to Disable the WatchDog function from resetting the CPU. **Value** / **Description** 1 – Enabled (default) 0 – Disabled
5	WatchDog Reset Status	When a Warm Reset is caused by the WatchDog Timer overflowing, the WatchDog Reset Status Bit Field is set to'1'. The status bit may be reset by a software write to the register changing that bit value as a '0'. **Value** / **Description** 1 – WatchDog Reset occurred 0 – WatchDog Reset has not occurred (default)
4	CPU BusError Enable	If the CPU BusError Enable is set, then if the CPU BusError Timer reaches its compare count and thus overflows, a BusError and an interrupt (if enabled) will be generated to the CPU core. This BusError is caused either by the CPU core taking too long or by the CPU core generating an undecodable address. See the BusError Exception Disable (bit 7) for more information. **Value** / **Description** 1 – Enabled (default) 0 – Disabled

Table 8.12 BusError Control Register Field Descriptions (Part 1 of 2)

Notes

Bits	Field Name	Description
3	IP BusTimeOut Enable	If the IP BusError Enable is set, then if the CPU BusError Timer reaches its compare count and thus overflows, an IP BusError and an interrupt (if enabled) will be generated to the IP Bus and if an on-chip peripheral module or the CPU core owns the CPU bus, a BusError will be generated to the CPU core. This BusError is caused either by DMA taking too long or by DMA generating an undecodable address. See the BusError Exception Disable (bit 7) for more information. Note that the IP Bus TimeOut value (in nsec) must be greater than the PCI Retry TimeOut multiplied by the TRDY TimeOut value (in nsec). . Value: 1 — Enabled (default); 0 — Disabled
2	CPU Source	If a CPU BusError is caused by the CPU BusError Timer overflowing, then the CPU BusError Status Bit Field is set. The status bit may be unset by a software write to the register with that bit value as a '0'. Value: 1 — CPU BusError occurred from a CPU transaction; 0 — CPU BusError did not occur (default)
1	IP Source	If an IP BusError is caused by the IP BusError Timer overflowing, then the IP BusError Status Bit Field is set. The status bit may be unset by a software write to the register with that bit value as a '0'. In the present RC32334 implementation, a CPU bus error also sets the IP BusError Status bit. Value: 1 — IP BusError occurred from an IP transaction 5 MHz; 0 — IP BusError did not occur (default)
0	Read/write Type	Value: 1 — BusError occurred on a read; 0 — BusError occurred on a write (default)

Table 8.12 BusError Control Register Field Descriptions (Part 2 of 2)

SysID Register

The SysID Register can be used by boot OS software to recognize the type of System Controller being used and to initialize it accordingly.The SysID is unique to each type of IDT System Controller. It can be used to differentiate between a system consisting of the discrete RC32364 CPU and RC32134 System Controller parts versus the fully integrated RC32334 part. The SysID Register can also be used to differentiate between any hardware silicon revision upgrade improvements that might occur in the future. For software users, the SysID is similar and can be used in conjunction with the CPU Core CP0 Processor Revision ID Register (PRId). For hardware and hardware debug systems, the SysID is also similar in concept and can be used in conjunction with the JTAG DEVICE ID register and the EJTAG DEVICEID register.

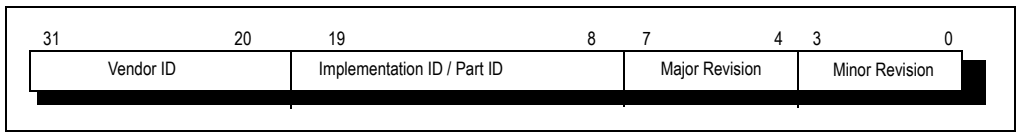


Figure 8.14 SysID Register Fields

Notes

Bits	Field Name	Description
31:20	Vendor ID	Vendor is IDT (0x000)
19:8	Implementation ID/Part ID	Part ID is RC32334 (0x002) Part ID is RC32332 (0x004)
7:4	Major Revision	Major revision: Rev Z silicon is (0x0) Rev Y silicon is (0x1) Reserved for future major revisions (0x2—0xF)
3:0	Minor Revision	Minor manufacturing revision number within a major revision. Base minor revision (0x0) 1st minor revision (0x1) Reserved for future minor revisions (0x2—0xF)

Table 8.13 SysID Register Field Descriptions

Notes

External Local Bus Interface

Notes

Introduction

As described at the beginning of Chapter 8, the RC32334 integrated processor is a logical integration of two components, a stand-alone CPU core previously implemented in the RC32364 device and a companion RC32134 system controller. This chapter describes the bus (called local bus) that is used to connect external devices to the RC32334. The local bus includes the following:

- *Separate Address and Data busses*
- *Control Signals (Chip Selects, Wait Signal, etc.)*
- *Debug Signals for Logic Analyses*

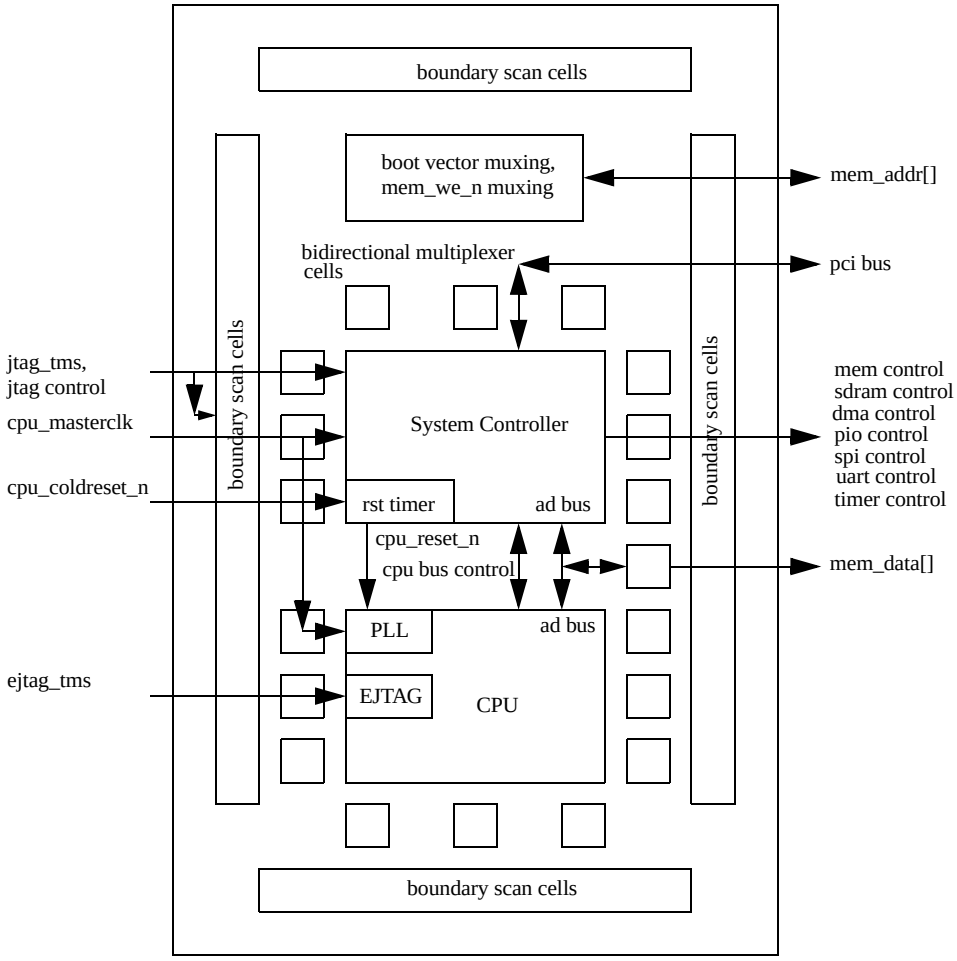


Figure 9.1 External Local Bus Interface Unit Block Diagram

Operation

The RC32334 Local Bus Interface Unit combines the internal busses from the CPU core and from the System Controller. The local address bus, mem_addr[25:2][1], is formed by the internal system controller latching the address on the RC32300 and then redriving it out. It also shifts the address signals appropri-

1. In the RC32332, this local address bus is mem_addr[22:2].

Notes

ately during SDRAM RAS cycles. mem_addr[1:0] are formed by multiplexing the CPU address signals [1:0] onto the mem_we_n signals. The RC32334 Local Bus Interface Unit also combines the internal data busses from the CPU and from the System Controller via a transceiver. The transceiver implementation does not carry internally tristated components, but uses multiplexing instead. Similarly, common signals between the CPU and System Controller internal components are combined.

Other output only signals, such as cpu_dt_r_n, can be an output from both the CPU and the System Controller. During CPU controlled cycles, it is driven from the CPU, and during System Controller controlled cycles, it is driven from the System Controller. During idle bus cycles, cpu_dt_r_n tristates on the local bus and thus requires an external pull-up resistor.

Similarly, jtag_tdo, the JTAG Output drives data from the System Controller Boundary Scan Register when the appropriate JTAG command is scanned via jtag_tms and drives data from the CPU EJTAG when the appropriate EJTAG command is scanned via ejtag_tms. The use of jtag_tdo assumes that both JTAG and EJTAG are not programmed simultaneously.

The reset boot mode vectors are formed by multiplexing inputs onto the appropriate CPU and System Controller inputs during cold reset.

Variable Port-Width Interface

The RC32334 supports a variable port-width interface. The technique used to determine the port width is a start-up and software mechanism that assigns attributes to a region of physical (not virtual) memory. The RC32334 reset-mode initialization interface supports the setting of the boot-prom port width.

To simplify design, the RC32334 elects to use the same data lines, for a given width of memory, regardless of memory byte-ordering (endianness[1]). Table 9.1 lists which byte lanes are used and Table 9.2, Table 9.3, and Table 9.4 list the data transfer sequences for 8-, 16-, and 32-bit port widths.

Port Width	Data Lines
8-bit	D(7:0)
16-bit	D(15:0)
32-bit	D(31:0)

Table 9.1 Port Width Assignments to Data Lines

Port Width	Transfer Size	Byte Address	Endianness	Data Lines
8-bit	1 byte	0, 1, 2, 3	Big	D(7:0)
8-bit	2 bytes	0, 2	Big	D(7:0) 2 times
8-bit	3 bytes	0, 1	Big	D(7:0) 3 times
8-bit	4 bytes	0	Big	D(7:0) 4 times
8-bit	16 bytes	0	Big	D(7:0) 16 times
8-bit	1 byte	0, 1, 2, 3	Little	D(7:0)
8-bit	2 bytes	1, 3	Little	D(7:0) 2 times
8-bit	3 bytes	3, 2	Little	D(7:0) 3 times
8-bit	4 bytes	3	Little	D(7:0) 4 times
8-bit	16 bytes	3	Little	D(7:0) 16 times

Table 9.2 Data Transfer Sequences for 8-bit Port Width

1. Little/big-endian byte ordering conventions are discussed in Chapter 1 of this manual.

Notes

Port Width	Transfer Size	Byte Address	Endianness	Data Lines
16-bit	1 byte	0, 2	Big	D(15:8)
16-bit	1 byte	1, 3	Big	D(7:0)
16-bit	1 byte	0, 2	Little	D(7:0)
16-bit	1 byte	1, 3	Little	D(15:8)
16-bit	2 bytes	0, 2	Big	D(15:0)
16-bit	2 bytes	0, 2	Little	D(15:0)
16-bit	3 bytes	0	Big	D(15:0), D(15:8)
16-bit	3 bytes	1	Big	D(7:0), D(15:0)
16-bit	3 bytes	0	Little	D(15:0), D(7:0)
16-bit	3 bytes	1	Little	D(15:8), D(15:0)
16-bit	4 bytes	0	Big	D(15:0) 2 times
16-bit	4 bytes	0	Little	D(15:0) 2 times
16-bit	16 bytes	0	Big	D(15:0) 8 times
16-bit	16 bytes	0	Little	D(15:0) 8 times

Table 9.3 Data Transfer Sequences for 16-bit Port Width

Port Width	Transfer Size	Byte Address	Endianness	Data Lines
32-bit	1 byte	0	Big	D(31:24)
32-bit	1 byte	1	Big	D(23:16)
32-bit	1 byte	2	Big	D(15:8)
32-bit	1 byte	3	Big	D(7:0)
32-bit	1 byte	0	Little	D(7:0)
32-bit	1 byte	1	Little	D(15:8)
32-bit	1 byte	2	Little	D(23:16)
32-bit	1 byte	3	Little	D(31:24)
32-bit	2 bytes	0	Big	D(31:16)
32-bit	2 bytes	2	Big	D(15:0)
32-bit	2 bytes	0	Little	D(15:0)
32-bit	2 bytes	2	Little	D(31:16)
32-bit	3 bytes	0	Big	D(31:8)
32-bit	3 bytes	1	Big	D(23:0)
32-bit	3 bytes	0	Little	D(23:0)
32-bit	3 bytes	1	Little	D(31:8)
32-bit	4 bytes	0	Big	D(31:0)
32-bit	4 bytes	0	Little	D(31:0)
32-bit	16 bytes	0	Big	D(31:0) 4 times
32-bit	16 bytes	0	Little	D(31:0) 4 times

Table 9.4 Data Transfer Sequences for 32-bit Port Width

Notes

Debug Signals

The RC32334 provides a set of debug signals for logic analyzer use. The four signals debug_cpu_ads_n, debug_cpu_ack_n, debug_cpu_dma_n and debug_cpu_i_d_n are used as reset mode bits during the assertion of cpu_coldreset_n as shown in Figure 19.9. These four signals begin driving from the RC32334 after cpu_coldreset_n de-asserts. debug_cpu_ads_n, debug_cpu_ack_n and debug_cpu_dma_n become valid 2 clocks after cpu_coldreset_n de-asserts. debug_i_d_n does not become valid until the first debug_ads_n asserts.

During a bus transaction, debug_cpu_ads_n will assert low for 1.0 clock. debug_cpu_ads_n asserts for both CPU generated transactions and for DMA generated transactions. Whenever debug_cpu_ads_n asserts, debug_cpu_dma_n will indicate the source of the transaction as being from the CPU or from DMA and whether the source is for an instruction or for data via the debug_cpu_i_d_n signal. During all DMA, debug_cpu_i_d_n will always indicate data as being the source.

Also when debug_cpu_ads_n asserts, mem_data[31:4] bus will contain the physical address of the quad-word block where the transaction is occurring. If the transaction is from DMA, then mem_data[3:2] will also indicate the word address from which the transaction is occurring.

If the transaction is from the CPU, then the mem_addr[3:2] lines must be used to determine the word address. Depending on the Address Latch Timing Register and Memory versus SDRAM CAS cycle occurring, the earliest strobe point for mem_addr[3:2] may vary. If an 08-bit wide or 16-bit wide transaction is performed via the Memory Controller, then address bits 1 and/or 0 may be taken off of mem_we_n bus.

After debug_cpu_ads_n asserts, 1.0 clock later and throughout the transaction, cpu_dt_r_n indicates whether the transaction is a write or when asserted, a read. Note that cpu_dt_r_n may assert earlier than 1.0 clock after debug_cpu_ads_n, especially during a CPU transaction, but definitely not during a DMA transaction.

Finally, after the appropriate number of internal wait-states has occurred, debug_cpu_ack_n will assert to indicate that data is being latched on a read, or that data is finished being transmitted on a write. Note that on a burst transaction, debug_cpu_ack_n will assert for each datum.

Notes

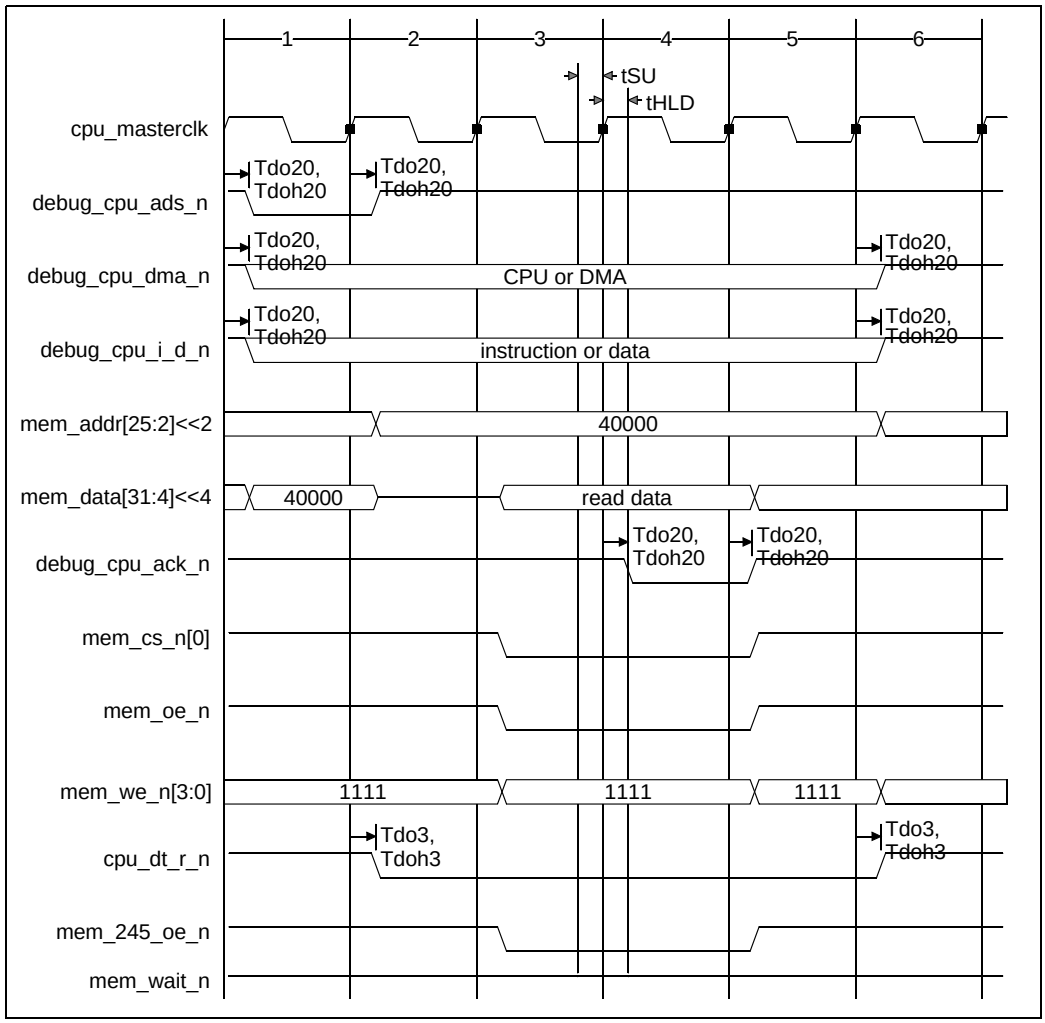


Figure 9.2 Debug Signals During a Read

In Figure 9.2, the debug signals are shown for a single word read, starting with the assertion of debug_cpu_ads_n. A 32-bit wide Memory Read of 1 word is shown. Note that the exact number of clocks between debug_cpu_ads_n and mem_cs_n[x] (sdram_cs_n[x]) may vary depending on such factors as the Address Latch Delay Setting and on the exact source of a CPU/DMA transaction.

Notes

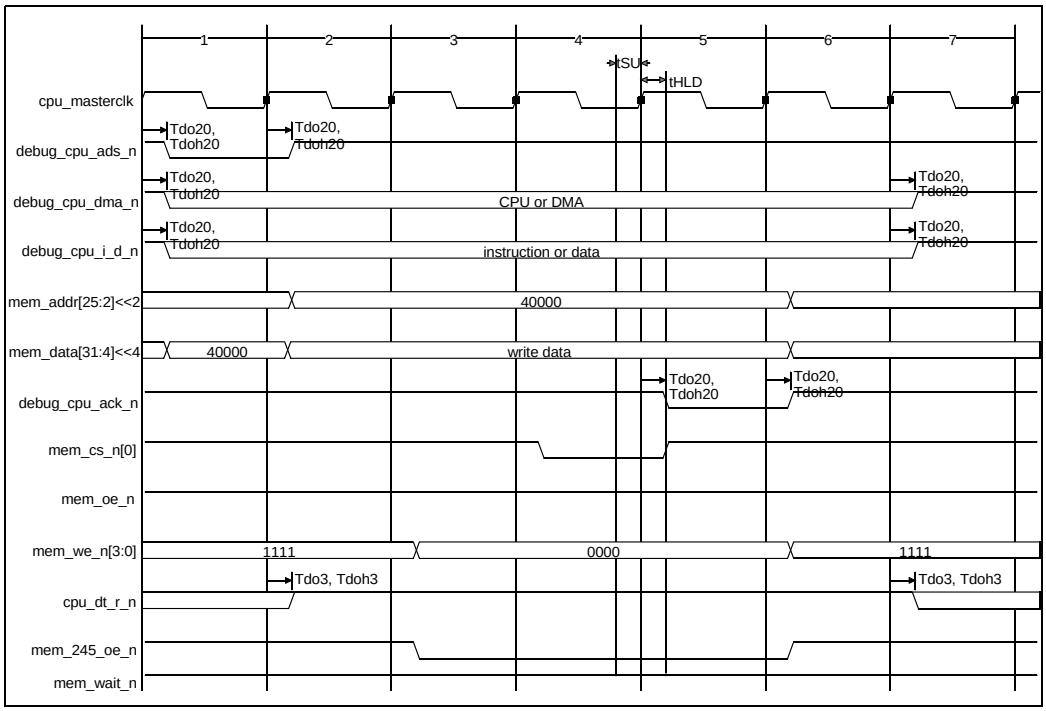


Figure 9.3 Debug Signals During a Write.

In Figure 9.3, the debug signals are shown for a single word write, starting with the assertion of debug_cpu_ads_n. A 32-bit wide Memory Write of 1 word is shown. Note that the exact number of clocks between debug_cpu_ads_n and mem_cs_n[x] (sdram_cs_n[x]) may vary depending on such factors as the Address Latch Delay Setting and on the exact source of a CPU/DMA transaction.

Memory Controller

Notes

Introduction

The RC32334 Memory Controller provides the control signals, address lines, and wait-state engine for interfacing RC32334 integrated processor to standard SRAM, PROM, FLASH and I/O. It also includes the boot-PROM interface. Six individual chip selects are also available, providing direct support of 8-, 16-, and 32-bit wide memory and I/Os.

The first two chip selects have highly configurable address ranges, allowing the selection of support for various memory types and widths. The last 4 chip selects have fixed address ranges. The RC32334 can interface directly to 8-/16-/32-bit boot-PROM width support, and has extra write protection for FLASH as well as a programmable number of wait-states for various speeds of memory and I/Os. For systems that require fast signalling with large loads, RC32334 also has controls for optional external data transceivers.

List of Features

- *Six chip selects*
  - *2 highly configurable address range decoders, 64KB to 64MB[1] each, anywhere within 4GB*
  - *4 fixed address range decoders, 32 MB each[1]*
  - *8-/16-/32-bit Memory and I/O support*
- *Selectable SRAM, two different I/Os, and Dual-Port protocol modes*
- *Programmable number of wait-states*
- *Single word read/write and burst read/write support*
- *External wait-state pin for debug Emulator Memory or Dual-Port memory*
- *FLASH default write protection*
- *Controls optional FCT245 transceiver*
- *8/16/32-bit boot-Prom support*

Block Diagram

This functional block diagram represents the Memory and I/O Control unit of the RC32334.

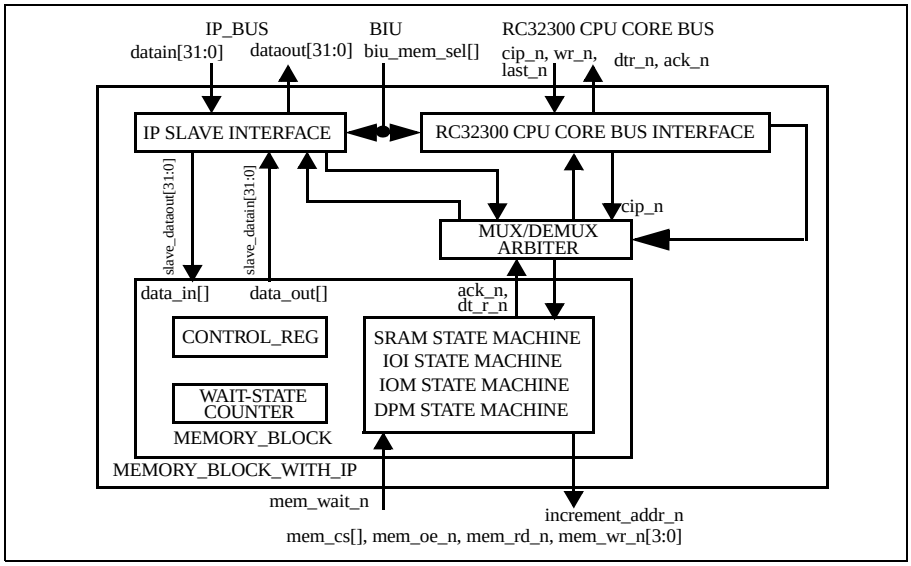


Figure 10.1 Block Diagram of RC32334 Memory Controller

1. To 8MB each for the RC32332.

Notes

Functional Overview

The RC32334 Memory Controller controls six memory or I/O peripheral regions. Each region has an associated chip select line (mem_cs_n[5:0]), but shares the rest of the control and address signals. To meet the system's needs, each memory and I/O region can be independently configured. Configuration options available for each bank are:

- *Address range (CS 0 and 1 only)*
- *Memory type*
- *Port width*
- *Wait-states and access speed, during read and write operations*

Each of these configuration options is enabled through the corresponding set of registers for each bank, which, at reset, will default to the user's previously software defined base configuration.

Memory Controller Operation

The Memory controller is activated when the CPU, DMA controller, or the PCI bridge issues a read or write transaction that is within the range of any of the six memory and I/O regions.

Integrated Processor Generated Transactions

When the controller issues a read or a write operation to an external memory or I/O peripheral, the memory controller looks at the address compares it with the address range of the six regions. If the address matches, the memory controller supplies the address bus, the various control signals, as well as the chip select for that region.

The appropriate configuration of the various registers enables the memory controller to assert the appropriate control signal and end the transaction at the specified time. The memory controller allows the controller to perform single or burst (up to 4 words) transfers from/to the memory and I/O peripherals for the read and write transfers.

Burst read transactions from the controller use a subblock ordering method, as shown in Figure 10.2. A subblock ordered transaction allows the system to define the order in which the data elements are retrieved.

```
Start Address[3:0] Burst Sequence
0                  (0,4,8,C)
4                  (4,0,C,8)
8                  (8,C,0,4)
C                  (C,8,4,0)
```

Figure 10.2 Subblock Ordered Burst Read Sequences

DMA Controller or PCI Bridge Generated Transactions

When the controller issues a read or a write operation to an external memory or I/O peripheral, the memory controller looks at the address and compares it with the address range of the six regions. If the address matches, the memory controller supplies the address bus with the various control signals as well as the chip select for that region. In this case, the data transfer occurs between the RC32334 and the external memory or I/O peripherals.

The RC32334 generates all control signals, per configuration of the various registers, and will also end the transaction at the specified time. During read transactions, the RC32334 samples the mem_data[31:0] bus to latch the data into the DMA or PCI FIFO. During write transactions, the RC32334 drives the data on the mem_data[31:0] bus from the DMA or PCI FIFO. From either the DMA controller or PCI bridge, the memory controller will support both read and write single and burst transactions[1].

[1] Burst transactions from DMA are linearly addressed.

Notes

Chip Selects

The Memory Controller contains 6 separate memory spaces, each having its own mem_cs_n output pin. The first two highly configurable mem_cs spaces occupy from 64K to 4GB of address space of which 64MB is externally addressable (due to the 26[1] address lines). The second through the fourth fixed memory chip select spaces occupy 32 MB of address space, all of which is externally addressable (due to the 26 address lines).[2] If the DMA or PCI bridge is not used to access a particular memory bank, an optional external FCT373 transparent latch can be used to extend the number of address bits for CPU accesses.

The address spaces that mem_cs_n[0] and mem_cs_n1[] decode are programmable. The Memory Controller uses the programmed information in the Base Address Registers, along with the size (64K to 8MB) of the given area as programmed in the Page Mask Registers, to setup the mem_cs spaces for banks 0 and 1.

This information is used to compare with the address asserted by the controller-BIU, DMA controller, or PCI bridge, to determine if that particular mem_cs_n area is being accessed for the current read or write operation. Each area supports single reads, burst reads, single writes, and burst writes. The port size of the data path (8, 16, 32-bit, or interleaved) of each area is also programmable through the appropriate control register.

Transceiver Control Interface

The Memory Controller provides transceiver output enables and write enables that are suitable for direct bus connection, or FCT245 transceivers. The selection of the type of memory is software programmable. FCT245s can be used for other banks, if the Boot PROM is also behind the FCT245s. The following are recommendations for system use:

- ***Transceivers and buffering in small to large systems***
  *In **small systems**, no glue logic is required. If the number of memory devices is eight or less, then address buffering is not required. If the number of memory and IO banks is eight or less, then data transceivers are also not required. In **medium systems** using more than eight memory chips, the address bus from RC32334 should be buffered using FCT244s. In general, the data bus does not need transceivers as often as the address bus needs buffering. Typically, for 8-bit chip devices, in each bank, there are four chips per address signal, but only one chip per data signal, yielding a 4-to-1 ratio. Typically, only the **largest systems** have more than eight banks of devices, at which point data transceivers are recommended.*
- ***Using slow-to-turn-off EPROMs in small to large systems***
  *In **small systems** using slow-to-turn-off-data EPROM or IO, the EPROM or IO data bus, for each bank, can be transceivered using FCT245s connected to mem_cs_n[5:0] and mem_245_dt_r_n(dir). Alternatively, the RC32334 bus turnaround (BTA) feature can be used to delay transactions after reads. In **medium systems** using slow-to-turn-off-data EPROM or IO, the EPROM and IO data bus can be transceivered, using a single bank of FCT245s connected to mem_245_oe_n(oe) and mem_245_dt_r_n(dir). In **large systems** using multiple banks of EPROM or IO, the EPROM or IO data bus for each bank can be transceivered using FCT245s connected to mem_cs_n[] and mem_245_dt_r_n(dir).*

Using 8- or 16-bit Boot PROMs

When using 8- or 16-bit boot PROMs,[3] the PROMs must use mem_addr[3:2] and mem_we_n[3:0] to provide the LSB system memory address bits. In the 16-bit case, the dynamic byte enables are provided via the RC32334's mem_we_n[3] and mem_we_n[0] signals. In the 8-bit case, the byte enable is provided by the RC32334's mem_we_n[0] signal.

1. 23 address lines for the RC32332.
2. The RC32334 has 26 address lines addressing a maximum 2^{26} (64) MB. The RC32332 has 23 address lines addressing a maximum 2^{23} (8) MB.
3. Note that 8-bit and 16-bit PROMs cannot use the RC32334 DMA to transfer data.

Notes

Note that the RC32334 coprocessor Port Width Control Register should be initialized by the boot PROM software for the non-boot regions of memory such as 32-bit wide DRAM regions.

Port Width	Pin Signals mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
DMA (32-bit)	mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
32-bit	mem_we_n[3]	mem_we_n[2]	mem_we_n[1]	mem_we_n[0]
16-bit	Byte High Write Enable	mem_addr[1]	Not Used (Driven Low)	Byte Low Write Enable
8-bit	Not Used (Driven High)	mem_addr[1]	mem_addr[0]	Byte Write Enable

Table 10.1 8- and 16-bit LSB Addresses and Write-Enable Connections

Wait-State Generator (WSG)

The WSG controls the speed of memory accesses to and from the internal Bus Interface Unit (BIU) controller, which includes the start time of a memory transaction until the first data are sent or received and the time between consecutive data on burst transactions. The signal called mem_wait_n can be used to override the WSG's programmed settings. When mem_wait_n is asserted, however, the actual action performed by the WSG depends upon when it is asserted, relative to the transaction. The mem_wait_n signal is also useful for accessing memories such as Dual-Port-type memory and other off-card memory where the acknowledge (Ack) signal must be connected to the mem_wait_n.

Address Decoding

Memory spaces are selectable up to 64MB[1] per channel. The first two Memory-I/O Channels have software selectability as to where and how much memory space the channel uses[2]. The remaining channels have fixed-size memory spaces. Within RC32334, the internal design is such that the address decoding and its registers is actually done with the RC32334-to-IP Bridge hardware. For readability reasons, the memory decode functionality is described in this chapter.

User Notes:

1. MEM/IO 0 space is used for reset boot ROMs typically starting at `0x1fc0_0000,` and thus is limited to a linear 4M from the boot reset address. If the ROM is wrapped around after 4M, then the lower portion of the address range can also be accessed.
2. MEM/IO spaces larger than 64MB require an external address latch, and in that case, can only be accessed via the controller (and not DMA or PCI).
3. MEM/IO spaces within the same controller physical region must have the same port width and BTA settings; for example, MEM/IO 2&3 and MEM/IO 4&5.
4. MEM/IO 1 space in this example uses an addressable region that may not be accessible in future RC32334 derivatives. If such a derivative is on the user's road map, MEM/IO 1 should be assigned to another area of memory, i.e., from `0x1f00_0000`.

1. 8MB for the RC32332.
2. Note that MEM/IO spaces within the same region must have the same port width and BTA settings, for example MEM/IO 2&3, and MEM/IO 4&5. Also, future channels may split the 0x1600_0000 to 0x 17FF_FFFF memory space into several more sections. Users can easily externally decode a chip select to expand the number of selectable devices, and through the use of an external decoder—such as a F138/F139 logic device—using the MSB system memory address bits, extra chip selects can be provided to like size/speed devices.

Notes

RC32334				
Physical Address Range		Max. Size	Region RC32300 CPU Core	Region Description
0000_0000	0FFF_FFFF	256MB	A,B,C,D	DRAM 0,1,2,3 (16MB typical)
1000_0000	11FF_FFFF	32MB	E	MEM/IO 2 (8MB typical)
1200_0000	13FF_FFFF	32MB	E	MEM/IO 3 (8MB typical)
1400_0000	15FF_FFFF	32MB	F	MEM/IO 4 (8MB typical)
1600_0000	17FF_FFFF	32MB	F	MEM/IO 5 (8MB typical)
1800_0000	1BFF_FFFF	64MB	G	RC32334 Internal Registers
1C00_0000	1FFF_FFFF	64MB	H	MEM/IO 0 (4MB typical)
2000_0000	23FF_FFFF	64MB	I	MEM/IO 1 (8MB typical)
4000_0000	5FFF_FFFF	512MB	J	PCI Memory Space 1 (256MB typical)
6000_0000	7FFF_FFFF	512MB	K	PCI Memory Space 2 (256MB typical)
8000_0000	FF1F_FFFF		O	Reserved, undecoded. If accesses are made to this region, the RC32334 will return a bus error.
FF20_0000	FF2F_0000	1MB	O	Reserved for RC32334 on-chip EJTAG Interface. The RC32334 does not decode this address and a bus error will be returned if accessed.

Table 10.2 RC32334 Typical Memory Map

Memory Type and Port-Width Size Support

Encoding the memory-type field with the values listed in Table 10.3 determines the bus interface timing to be supported by the memory controller. The Port Size field values listed in Table 10.4 determine the width of the memory or I/O port. One of the four memory types described below must be selected, as shown in Table 10.3:

- ***FLASH/SRAM*** *This is the default memory type. The read data is primarily controlled by the mem_oe_n signal which enables the read data back onto the mem_data[] bus on a continuous basis. Write data is primarily controlled by the chip select, mem_cs_n[x], which is used as the write strobe.*
- ***IOI (Intel-Type I/O)*** *This I/O type uses separate read and write strobes to signal valid data. Customarily, this I/O type uses single word accesses.*
- ***IOM (Motorola-Type I/O)*** *This I/O type uses a common data strobe, mem_cs_n[] and uses mem_oe_n or mem_we_n[] as write/read_n or read/write_n status lines. Customarily, this I/O type uses single word accesses.*
- ***DPM (Dual-Port Memory)*** *The read data is primarily controlled by the mem_oe_n signal which enables the read data back onto the mem_data[] bus on a continuous basis. Write data is primarily controlled by the write enables, mem_we_n[3:0], which are used as the write strobes. The external wait-state pin, mem_wait_n when asserted, resets the internal Wait-State generator back to its initial value, such that when mem_wait_n is de-asserted, the internal wait-state count start over giving a full length transaction from that point in time.*

  Note: The Write Protect field and the Port Width Size field of the Memory control Register for each bank must be setup before any writes to that bank occur. The Write Protect field defaults to protected, thus it must be unset to first issue a write. During write protection, the chip select remains de-asserted.

Notes

When choosing a Dual-Port Memory, the semaphore or interrupt version can be useful for data buffer streams. If the busy version is used, the external mem_wait_n signal (see Table 10.11 for signal definitions) can be used for busy-signal support by using the Dual-Port Memory Type. In the Dual-Port Memory Type, if the busy pin is asserted, then the Memory Controller's wait-state counter resets to 0.

During the first clock, the busy pin is always ignored; thus, for seamless integration, a Dual-Port Memory part—where busy_n becomes valid between 1 and 2 clocks—must be selected. If a slower part were to be selected, application specific wait-state programming and external delay logic would be necessary, to mask out the indeterminate busy flag for the first few clocks.

Value	Action
11	Dual Port **Note:** If the Dual-Port mode is selected, all other memory types will ignore the mem_wait_n pin. This prevents inadvertent lockups from matching addresses during non-Dual-Port transactions.
10	M-type I/O = Motorola type I/O
01	I-Type I/O = Intel type I/O
00	Flash/SRAM (default)

Table 10.3 Memory Type Field Values and Actions

Dual-Port Memory type differs from Flash/SRAM Memory type in that:

- *mem_wait_n is handled differently*
- *writes use mem_we_n[3:0] controlled writes instead of mem_cs_n[x] controlled writes*

Dual-Port Memory Type reads drive the address, chip select, and output at the same time. Burst read transfers alter the address on subsequent data. Thus, if the external Dual-Port Memory is fast enough, true zero wait-state burst reads will be able to occur. Dual-Port Memory type writes also drive the address, data and chip select, but delay the assertion of mem_we_n[]. After the programmed number of wait-states, mem_we_n[] is de-asserted and the address, data, and chip select are held for 1 clock. As such, burst writes require a minimum of 3 clocks for each data burst.

Port-Width Size

Non-Interleaving, non-expandable 32-/16-/8-bit support for Bank 0 or Bank 1 can be 16- or 8-bit, but it might not be physically contiguous with Bank 0, unless it is at least 64Kx8. Note that in the former case, the TLB and the RC32334 integrated processor can be used to make physical memory virtually contiguous for linearly addressed software applications.

In the SRAM mode, 16-bit ports require that the Write Enable pins mem_we_n[3] and mem_we_n[0] be connected to the most and least significant bytes, respectively. Also, in the 16-bit mode, SRAM mode multi-byte writes will delay the subsequent assertion of mem_cs_n[] by one clock from the normal 32-bit or 8-bit cases, to allow the mem_we_n[3:0] signals to setup during burst writes.

As in the case of Dual-Port Memory, 8-bit ports may require that the Write Enable pin of the memory device be connected to the RC32334 pin, mem_we_n[0]. When in this mode, the memory controller asserts mem_we_n[0] on writes. In 8-bit mode, mem_we_n[2:1] serve as address bits 1 and 0.

Value	Action
11	Reserved
10	32-bit Port Width Size Writes (default)
01	16-bit Port Width Size Writes
00	8-bit Port Width Size Writes

Table 10.4 Port Width Size Field Values and Actions

Notes

Depending on the Port-Width size selected, the byte enable handling on writes will differ: 16-bit Port Width memories split a single word write into two mini-burst 16-bit data segments. In this case, the 16-bit Port Width mode will ensure that mem_we_n[3] and mem_we_n[0] are de-asserted on the first data segment to provide data hold time. 8-bit Port Width memories always assert mem_we_n[0] on each data word.

I/O Width Support

Because the RC32334 does not directly support byte enables on reads (it could be done externally), 32-bit I/O word-aligned devices are strongly recommended. 8- and 16-bit devices should be word-aligned, such that the MSB bits [31:8] and [31:16] are not used independent from endianness. Burst or mini-burst accesses are not recommended, although they will complete with an implementation specific method that does not necessarily meet any particular device's burst protocol or command recovery period (BTA period).

Programmable Wait-State Generator

A software programmable register (see Table 10.11 on page 10-11) allows selection of the number of wait-states from between 0 and 31 for reads versus writes. Data within bursts have identical settings to reads versus writes. According to the nature of each memory type's protocol, a minimum number of wait-states for asynchronous transfer types is required, as shown in Table 10.5.

Memory type	Transfer type	Minimum Wait-state requirements
sram	read	0
sram	write (32/8-bits)	2
sram	write (16-bits)	2 + (1 clk automatically inserted between data)
ioi	read	1 + (2 clks automatically inserted between data)
ioi	write	2 + (1 clk automatically inserted between data)
iom	read	1 + (2 clks automatically inserted between data)
iom	write	2 + (1 clk automatically inserted between data)
dpm	read	0
dpm	write	2

Table 10.5 Minimum Wait-State Settings

External Wait-State Behavior

On **SRAM, IOI (I/O Intel type), and IOM (I/O Motorola type) accesses**, the internal wait-state generator ignores the mem_wait_n signal until its last internal wait-state. At that time, users can add an arbitrary number of additional external wait-states. The last internal wait-state corresponds to the clock before cpu_ack_n would have asserted, thus mem_wait_n must be asserted any time before the final clock cycle of the transaction occurs. If the channel is using 0 wait-states internally, the first data cannot be stopped unless mem_wait_n is left asserted before the memory transaction begins.

On **Dual-Port accesses**, if mem_wait_n is always ignored during the first clock of a dpm transaction, it allows the dpm time to generate a valid busy_n signal. On subsequent clocks, mem_wait_n is internally synchronized for metastability by delaying it one clock, and the internal wait-state counter is then reset to 0 to restart the dpm transaction. If any channel uses the dpm mode, then mem_wait_n is automatically ignored during SRAM, IOI, and IOM accesses. Note that only 1 dpm may use mem_wait_n, unless external provisions are made to ensure that a dpm address match does not occur on other banks.

Typically, the RC32334 BTA register is also set up before switching to any bank besides the Boot PROM. The RC32334 BTAs are set to their maximum of 3 clocks and most memories can be set to a minimum of 1 clock (a Trecovery clock is always inserted, in addition to any BTA clocks).

Notes

Bus Error Recovery

The Memory Controller gracefully aborts on a bus error and bus time-out. Both bus errors and bus time-outs latch the present physical address into the BusError Address Register. This implies that the bus error controller can only assert cpu_buserr_n up until the first cpu_ack_n would be returned.

On a bus error, the memory controller ignores the mem_wait_n signal and returns cpu_ack_n to the CPU core, at the normal times indicated by the WSG until the transaction is complete. At the first cpu_ack_n, cpu_buserr_n is then asserted. A bus time-out may occur at any time during the bus transaction, even after the first ack_n has been returned. If the time-out occurs, the bus time-out interrupt is asserted, then the mem_wait_n is ignored and the transaction continues.

Signal Descriptions

Table 10.6 describes the signals used in RC32334's memory controller transactions. The list of memory controller registers is provided in Table 10.7, with register fields and their descriptions listed in Table 10.11.

<table>
<tr><th>Name</th><th>Type</th><th></th><th>Description</th></tr>
<tr><td colspan="4">Memory/I/O Controller</td></tr>
<tr><td>mem_addr[25:2]¹

mem_addr subsets
<table><tr><td>mem_addr[22:20]</td><td>I/O</td><td>reset_boot_mode[1:0]
reset_pci_host_mode</td></tr><tr><td>mem_addr[19:17]</td><td>I/O</td><td>reset mode bits</td></tr><tr><td>mem_addr[15:2]</td><td>I/O</td><td>sdram_addr[15:2]</td></tr></table></td><td>I/O</td><td>Not applicable</td><td>Memory Address Bus
These signals provide the Memory or DRAM address, during a Memory or DRAM bus transaction. During each word data, the address increments either in linear or sub-block ordering, depending on the transaction type. The table below indicates how the memory write enable signals are used to address discrete memory port width types.

<table><tr><th>Port Width</th><th>Pin Signals
mem_we_n[3]</th><th>mem_we_n[2]</th><th>mem_we_n[1]</th><th>mem_we_n[0]</th></tr><tr><td>DMA (32-bit)</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr><tr><td>32-bit</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr><tr><td>16-bit</td><td>Byte High Write Enable</td><td>mem_addr[1]</td><td>Not Used (Driven Low)</td><td>Byte Low Write Enable</td></tr><tr><td>8-bit</td><td>Not Used (Driven High)</td><td>mem_addr[1]</td><td>mem_addr[0]</td><td>Byte Write Enable</td></tr></table>
mem_addr[22] Alternate function: reset_boot_mode[1].
mem_addr[21] Alternate function: reset_boot_mode[0].
mem_addr[20] Alternate function: reset_pci_host_mode.
mem_addr[19] Alternate function: modebit [9].
mem_addr[18] Alternate function: modebit [8].
mem_addr[17] Alternate function: modebit [7].
mem_addr[15] Alternate function: sdram_addr[15].
mem_addr[14] Alternate function: sdram_addr[14].
mem_addr[13] Alternate function: sdram_addr[13].
mem_addr[11] Alternate function: sdram_addr[11].
mem_addr[10] Alternate function: sdram_addr[10].
mem_addr[9] Alternate function: sdram_addr[9].
mem_addr[8] Alternate function: sdram_addr[8].
mem_addr[7] Alternate function: sdram_addr[7].
mem_addr[6] Alternate function: sdram_addr[6].
mem_addr[5] Alternate function: sdram_addr[5].
mem_addr[4] Alternate function: sdram_addr[4].
mem_addr[3] Alternate function: sdram_addr[3].
mem_addr[2] Alternate function: sdram_addr[2].</td></tr>
</table>

Table 10.6 Memory Controller Pin Descriptions (Part 1 of 2)

Notes

Name	Type		Description
mem_cs_n[5:0]	O	Not applicable	**Memory Chip Select Negated** Recommend an external pull-up. Signals that a Memory Bank is actively selected.
mem_oe_n	O	Not applicable	**Memory Output Enable Negated** Recommend an external pull-up. Signals that a Memory Bank can output its data lines onto the cpu_ad bus.
mem_we_n[3:0]	O	Not applicable	**Memory Write Enable Negated Bus** Signals which bytes are to be written during a memory transaction. Bits act as Byte Enable and mem_addr[1:0] signals for 8-bit or 16-bit wide addressing.
mem_wait_n	I		**Memory Wait Negated** Requires an external pull-up. SRAM/IOI/IOM modes: Allows external wait-states to be injected during last cycle before data is sampled. DPM (dual-port) mode: Allows dual-port busy signal to restart memory transaction. Alternate function: sdram_wait_n.
mem_245_oe_n	O	Not applicable	**Memory FCT245 Output Enable Negated** Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to a memory or I/O bank.
mem_245_dt_r_n	O	cpu_dt_r_n	**Memory FCT245 Direction Xmit/Rcv Negated** Recommend an external pull-up. Alternate function: cpu_dt_r_n.

Table 10.6 Memory Controller Pin Descriptions (Part 2 of 2)

1. mem_addr[22:2] for the RC32332.

Register Definitions

Table 10.7 provides the physical hardware address locations of the Memory Controller registers, which include Memory Base and Mask registers for Banks 1:0 and the Control registers for Banks 5:0. Fields of the Bank and Mask registers are shown in Figure 10.3 and Figure 10.4.

Details on these fields are provided in Table 10.9 and Table 10.10. The Memory Control register information is provided in Figure 10.5 and Table 10.11.

Physical Address	Register Descriptions
1800_0080	Memory Base Address Register for Bank 0
1800_0084	Memory Base Mask Register for Bank 0
1800_0200	Memory Control Register for Bank 0
1800_0088	Memory Base Address Register for Bank 1
1800_008C	Memory Base Mask Register for Bank 1
1800_0204	Memory Control Register for Bank 1
1800_0208	Memory Control Register for Bank 2
1800_020C	Memory Control Register for Bank 3
1800_0210	Memory Control Register for Bank 4
1800_0214	Memory Control Register for Bank 5

Table 10.7 List of Memory Control Registers

Notes

Memory MSB Base Address Register for Banks 1:0

The base address registers are used to determine the starting location of a particular chip select. There are six pairs of MSB and LSB registers, a pair for each Memory Chip Select (MemCS). Each pair of memory base address registers is concatenated onto an internal 32-bit register and refers to the most significant 16 address bits and the least significant 16 address bits. Unused bits are always read as '0'.

Typically, if two banks of PROM/SRAM are used, the larger bank is placed in the Bank 0 address space and the smaller bank is placed in the Bank 1 address space. This arrangement allows a contiguous address space for the two combined banks.

The default value for bank 0 base address register [1800_0080] is 0x1FCO_0000 when the RC32334 is programmed in the standard boot mode. The default is 0xFFCO_0000 when programmed with the reset vector PCI_boot mode or the Non_boot mode. The default value for bank1 base address register [1800-0088] is 0x 2000_0000.

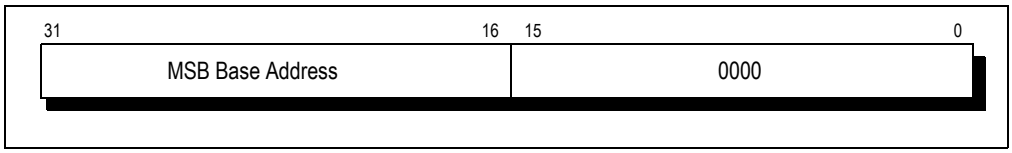


Figure 10.3 Memory Base Address Register for Banks 1:0

Internal Group	Base Address
mem_cs_n[0]	Default is 1FCO_0000 except if in PCI-boot or Non-boot mode where the default is FFCO_0000
mem_cs_n[1]	Default is 2000_0000
mem_cs_n[2]	Hard-coded to 1000_0000
mem_cs_n[3]	Hard-coded to 1200_0000
mem_cs_n[4]	Hard-coded to 1400_0000
mem_cs_n[5]	Hard-coded to 1600_0000

Table 10.8 Internal Chip Select Base Addresses

Memory MSB Bank Mask Registers for Banks 1:0

The Bank mask registers are used to determine the address bits in the base address that are to be used for comparing whether a chip select is to be activated. Unused bits are always read as '0'. The internal grouping of the six chip selects are as listed in Table 10.9.

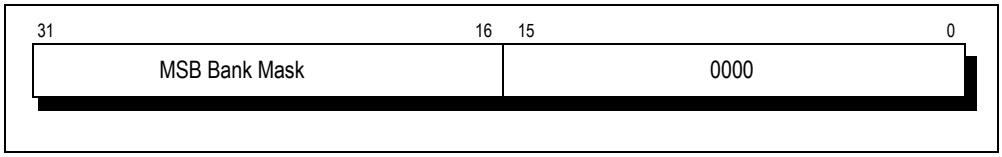


Figure 10.4 Memory Bank Mask Register for Banks 1:0

Notes

Internal Group	Chip-Select Mask Bit Activation
mem_cs_n[0]	Default is FFFF_0000
mem_cs_n[1]	
mem_cs_n[2]	Effective value is hard-coded as FE00_0000
mem_cs_n[3]	
mem_cs_n[4]	
mem_cs_n[5]	

Table 10.9 Internal Chip Select Grouping

Value	Action
1	Bit is used in address comparison
0	Bit is masked out

Table 10.10 Memory Mask Field Definitions and Values

Memory Control Register for Banks 5:0

Systems with multiple memory/IO banks can have all banks behind a FCT245 transceiver bank or they can have all banks on the CPU local bus. Systems may be “mixed” such that the boot memory Bank 0 is behind the FCT245 bank and other memory/IO banks can reside either behind the FCT245 transceiver bank or on the local CPU bus.

Note: For each bank, the Write Protect field (bit 12) and the Port Width Size field (bits 11:10) of the Memory Control register must be setup before any writes to that bank occur. The Write Protect field defaults to protected and must be unset prior to the first write issued. During write protection, the chip select remains de-asserted.

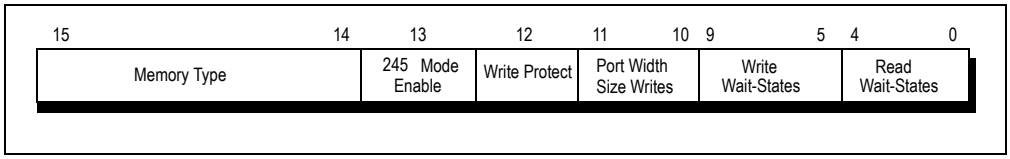


Figure 10.5 Memory Control Register Channel 5:0

Bit	Name	Description
15:14	Memory Type	See below

Value	Description
11	Dual Port **Note:** If the Dual-Port mode is selected, all other memory types will ignore the mem_wait_n pin. This prevents inadvertent lockups from matching addresses during non-Dual-Port transactions.
10	M-type I/O (Motorola type)
01	I-Type I/O (Intel type)
00	Flash/SRAM **(default)**

Table 10.11 Memory Controller Register Field Descriptions, Channels 5:0 (Part 1 of 2)

Notes

Bit	Name	Description
13	mem_245_oe_n Enable Field	The mem_245_oe_n Enable Field controls whether the mem_245_oe_n transceiver enable can assert or not. If it is disabled, then the memory bank for this channel can reside on the local CPU bus rather than behind a transceiver bank. **Value** / **Description** 1 — mem_245_oe_n enabled for this channel bank **(default)** 0 — mem_245_oe_n disabled for this channel bank such that it can reside on the CPU local bus
12	Write Protect Field	**Value** / **Description** 1 — mem_cs_n not asserted on writes (**default**) 0 — mem_cs_n asserts during writes (typical setting)
11:10	Port Width Size Writes	The function of this field is discussed in the section titled “Port-Width Size” on page 10-6. **Value** / **Size** 11 — Reserved 10 — 32-bit port width size writes **(default)** 01 — 16-bit port width size writes 00 — 8-bit port width size writes
9:5	Write Wait-States	This software programmable register allows selection of the number of wait-states for writes from between 0 and 31. Each memory type’s protocol requires a minimum number of wait-states. As shown, the default value is 31. **Value** / **Wait-States** 0 - 31 — Refer to Table 10.4 for transfer type requirements. The **default value** is **31**.
4:0	Read Wait-State	This software programmable register allows selection of the number of wait-states for reads from between 0 and 31. Each memory type’s protocol requires a minimum number of wait-states. As shown, the default value is 31. **Value** / **Wait-States** 0 - 31 — Refer to Table 10.4 for transfer type requirements. The **default value** is **31**.

Table 10.11 Memory Controller Register Field Descriptions, Channels 5:0 (Part 2 of 2)

Timing Diagrams

The timing of various memory and peripheral read and write operations is shown in the diagrams that follow. These diagrams include timing representations for both single and burst transfers. An operational overview description is provided before each diagram.

Figure 10.6 shows an SRAM-type single word memory read with 1 internally generated wait-state. Note that both the chip select, mem_cs_n[0], and the output enable, mem_oe_n, signals primarily determine the read access time for the data from the SRAM. Note that additional internally generated wait-states will repeat state 4. (Type=00, 245=1, WP=0, PW=10, WWS=2, RWS=1).

Notes

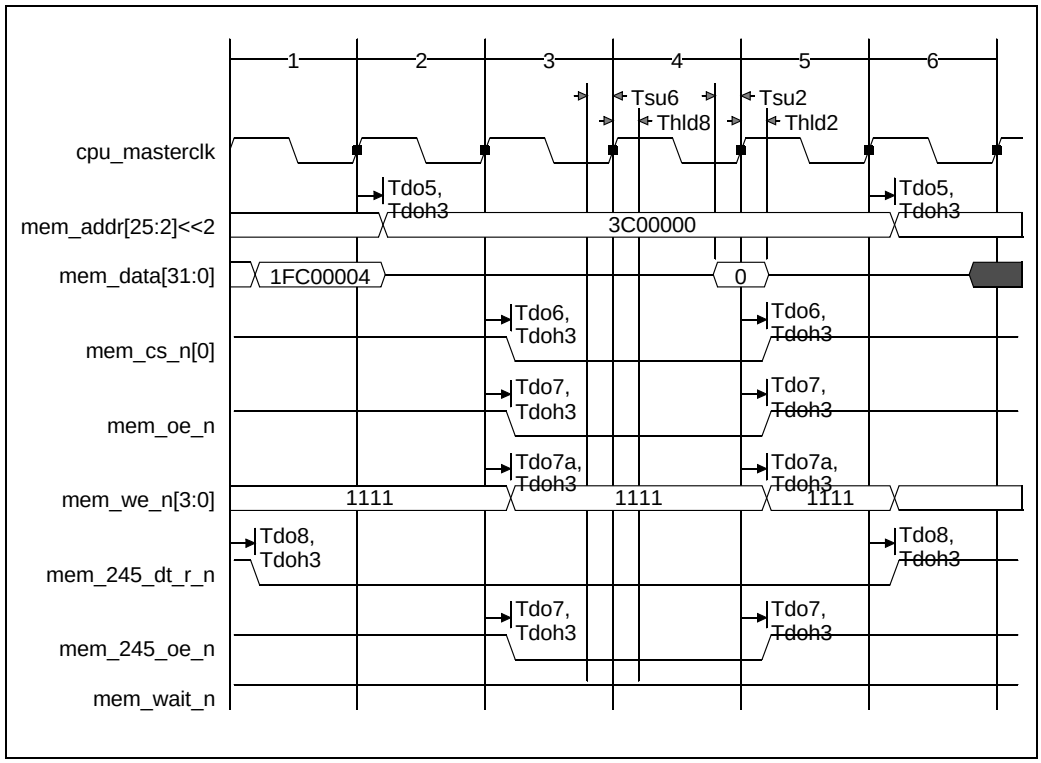


Figure 10.6 Single Word SRAM Read Transaction

Notes

Figure 10.7 shows an SRAM-type single word memory read with 1 internally generated wait-state and then 1 externally generated wait-state as indicated by mem_wait_n asserting. Note that if the memory controller were programmed such that 2 or more internally generated wait-states occurred, mem_wait_n is ignored until the final wait-state occurs the clock before where debug_cpu_ack_n would assert. (Type=00, 245=1, WP=0, PW=10, WWS=2, RWS=1).

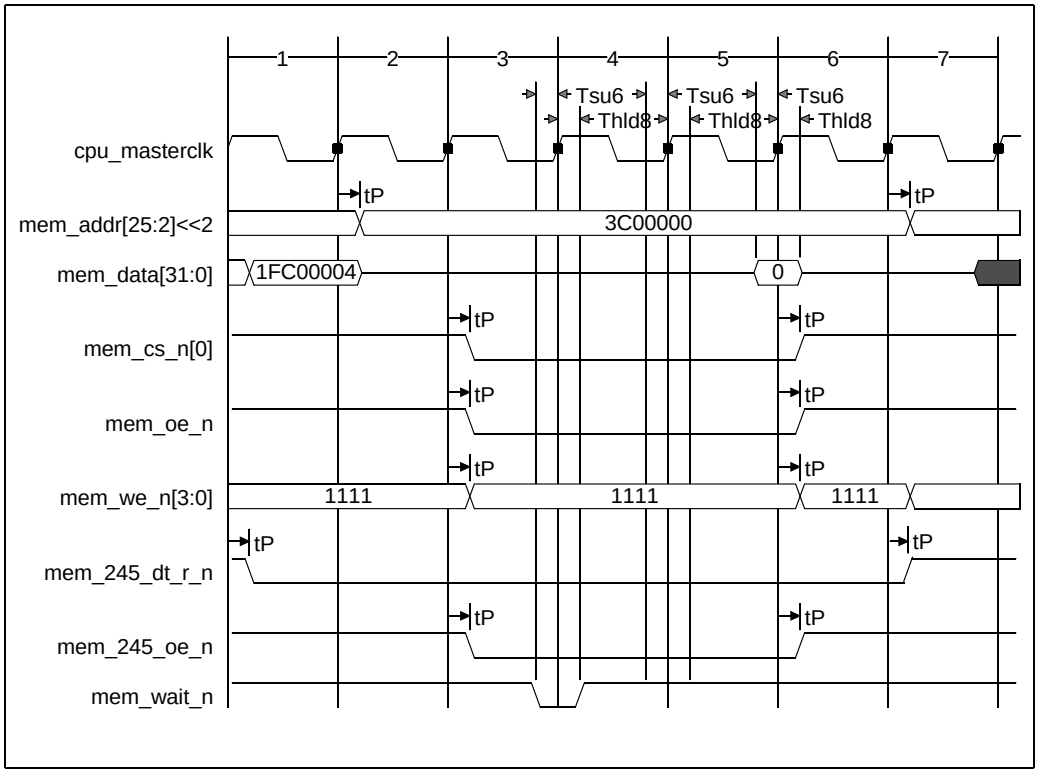


Figure 10.7 Single Word SRAM Read Transaction with Wait-State

Notes

Figure 10.8 shows an SRAM-type single word memory write with 2 internally generated wait-states. Note that the write enables, mem_we_n[3:0], are used as status lines, while the chip select, mem_cs_n[0], is used as the primary write strobe. Also note that additional internally generated wait-states will repeat state 4, so that mem_cs_n[0] is continuously asserted during internal wait-states. (Type=00, 245=1, WP=0, PW=10, WWS=2, RWS=1).

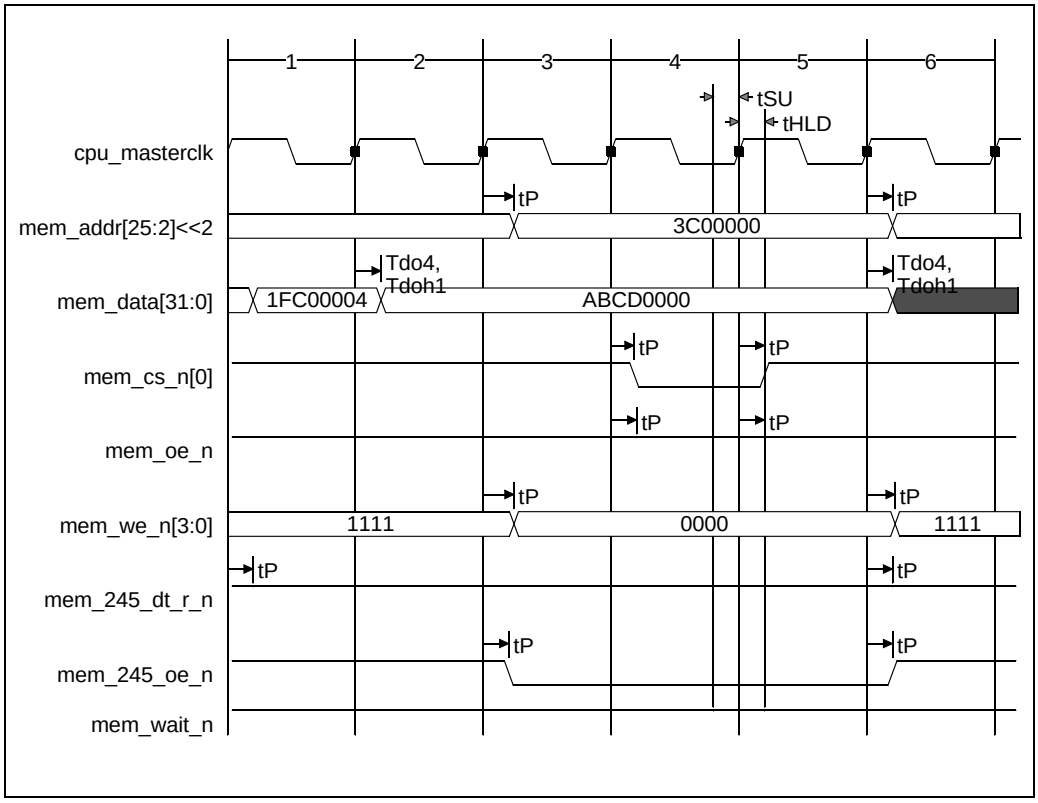


Figure 10.8 Single Word SRAM Write Transaction

Notes

Figure 10.9 shows an SRAM-type single word memory write with 3 internally generated wait-states and then 1 externally generated wait-state as indicated by mem_wait_n asserting. This case provides 1 more wait-state beyond the required minimum of 2 wait-states. Note that if the memory controller were programmed such that 3 or more internally generated wait-states occurred, mem_wait_n is ignored until the final wait-state occurs where debug_cpu_ack_n would assert. (Type=00, 245=1, WP=0, PW=10, WWS=3, RWS=2).

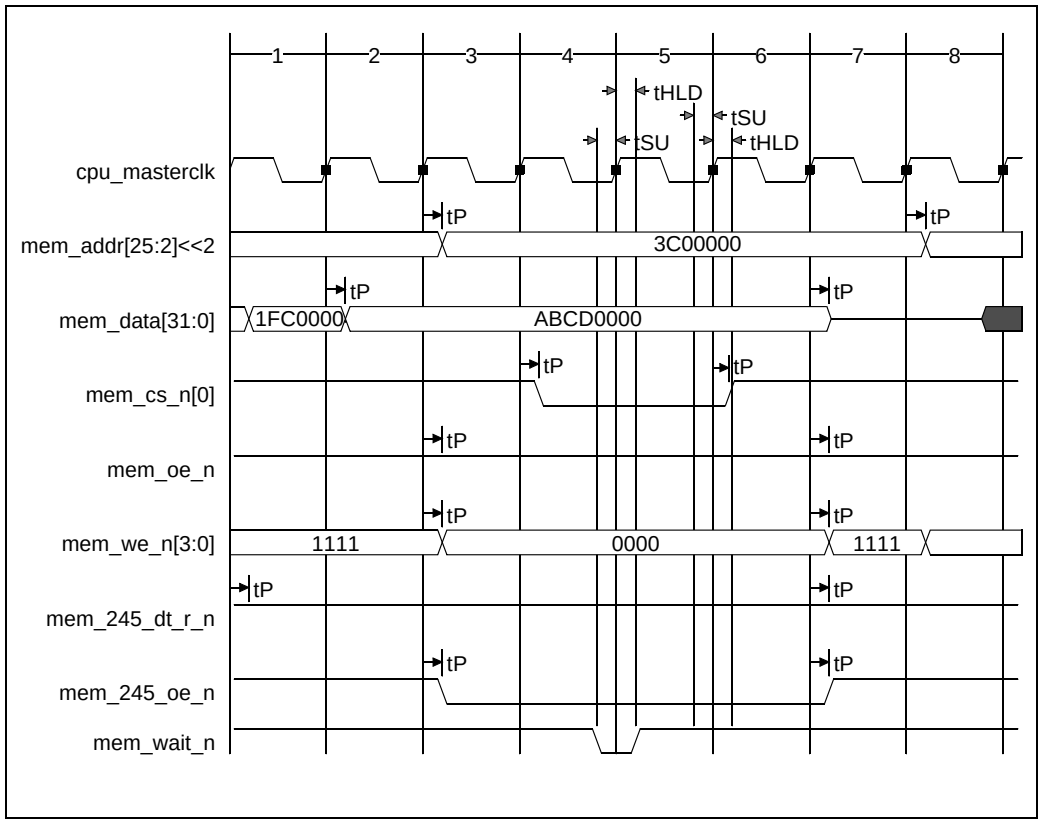


Figure 10.9 Single Word SRAM Write Transaction with Wait-State

Notes

Figure 10.10 shows an SRAM-type four word memory burst read with 1 internally generated wait-state on each data. Note that both the chip select, mem_cs_n[0], and the output enable, mem_oe_n, primarily determine the read access time for the data from the SRAM. (Type=00, 245=1, WP=0, PW=10, WWS=2, RWS=1).

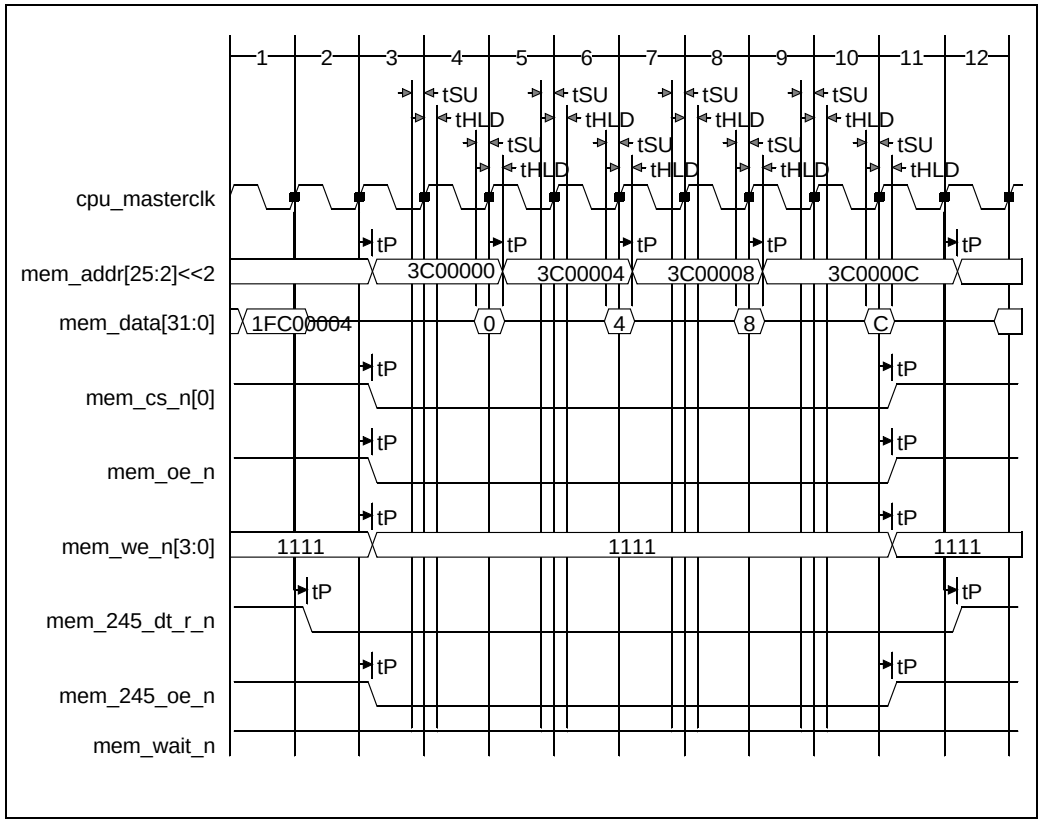


Figure 10.10 Quad Word Burst Read SRAM Transaction

Notes

Figure 10.11 shows an SRAM-type four word memory burst write with two internally generated wait-states. Note that the writes enables, mem_we_n[3:0], are used as status lines, while the chip select, mem_cs_n[0], is used as the primary write strobe. Note that if the access is to a 16-bit port-width, then an extra cycle (not shown) is automatically inserted between each datum, such that the write enables, mem_we_n[] can change dynamically for each halfword with both 1 clock setup and hold relative to the chip select asserting and de-asserting. (Type=00, 245=1, WP=0, PW=10, WWS=2, RWS=1).

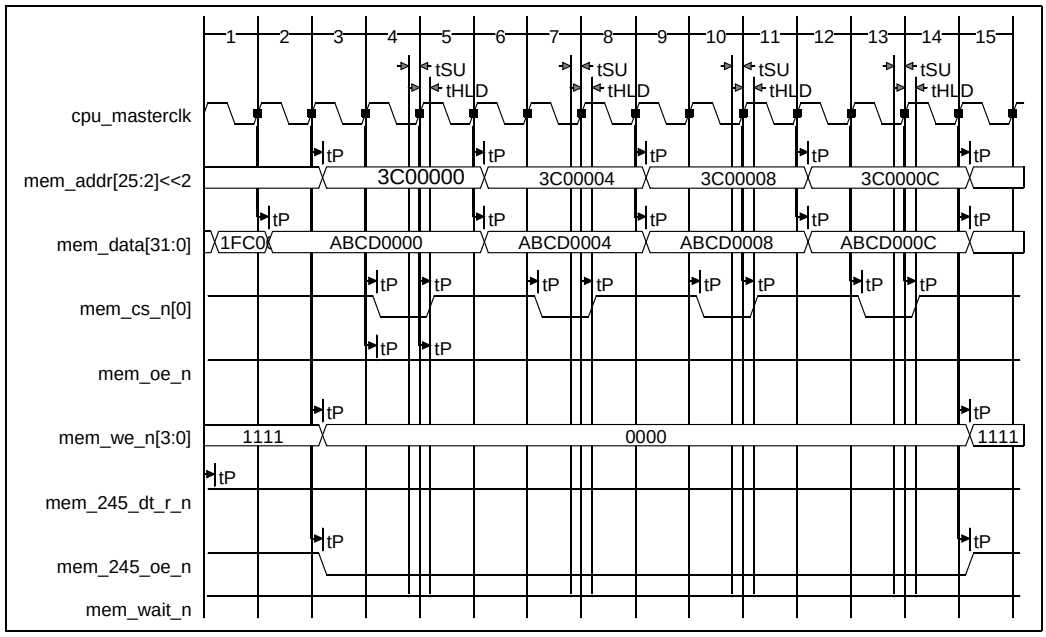


Figure 10.11 SRAM 4 Word Burst Write

Figure 10.12 shows an SRAM-type tri-byte mini-burst 16-bit port width write with 2 internally generated wait-states. Note that the second assertion of mem_cs[] is delayed one clock to allow mem_we_n[] time to setup. (Type=00, 245=1, WP=0, PW=10, WWS=2. RWS=1).

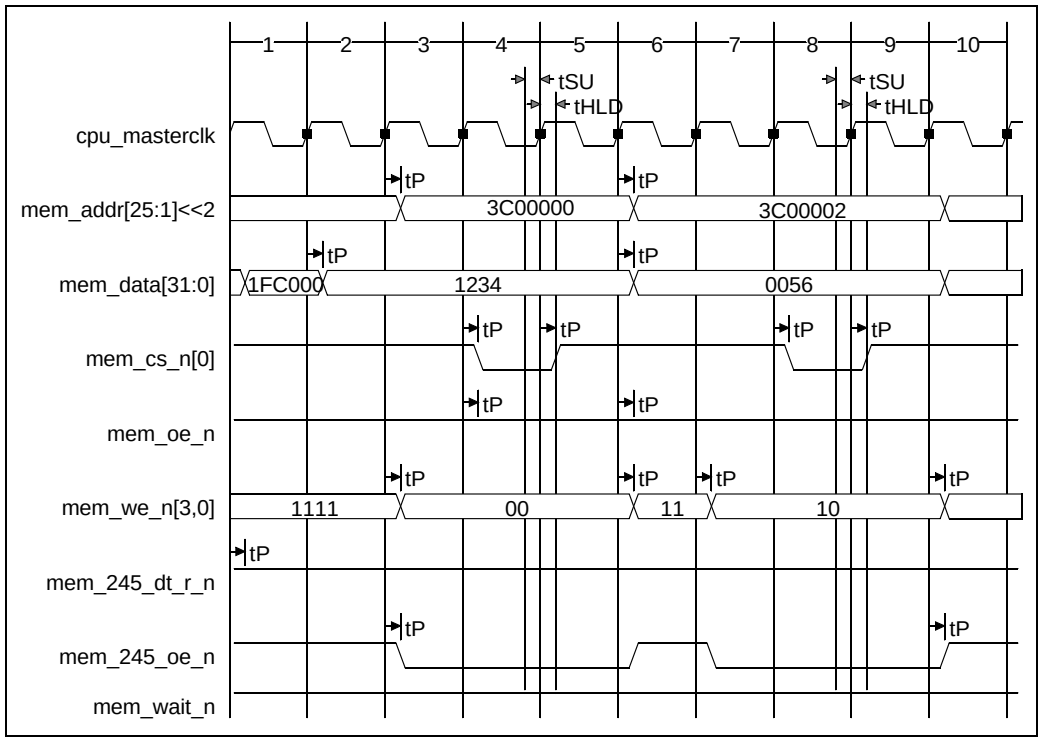


Figure 10.12 Tri-byte 16-bit SRAM Write Transaction

Notes

Figure 10.13 shows an IOI-type single word memory read with 1 internally generated wait-state. Note that the read output enable, mem_oe_n provides the read data strobe.

Note that additional internally generated wait-states will repeat state 4, so that mem_oe_n is continuously asserted during internal wait states. (Type=01, 245=1, WP=0, PW=10, WWS=2, RWS=1).

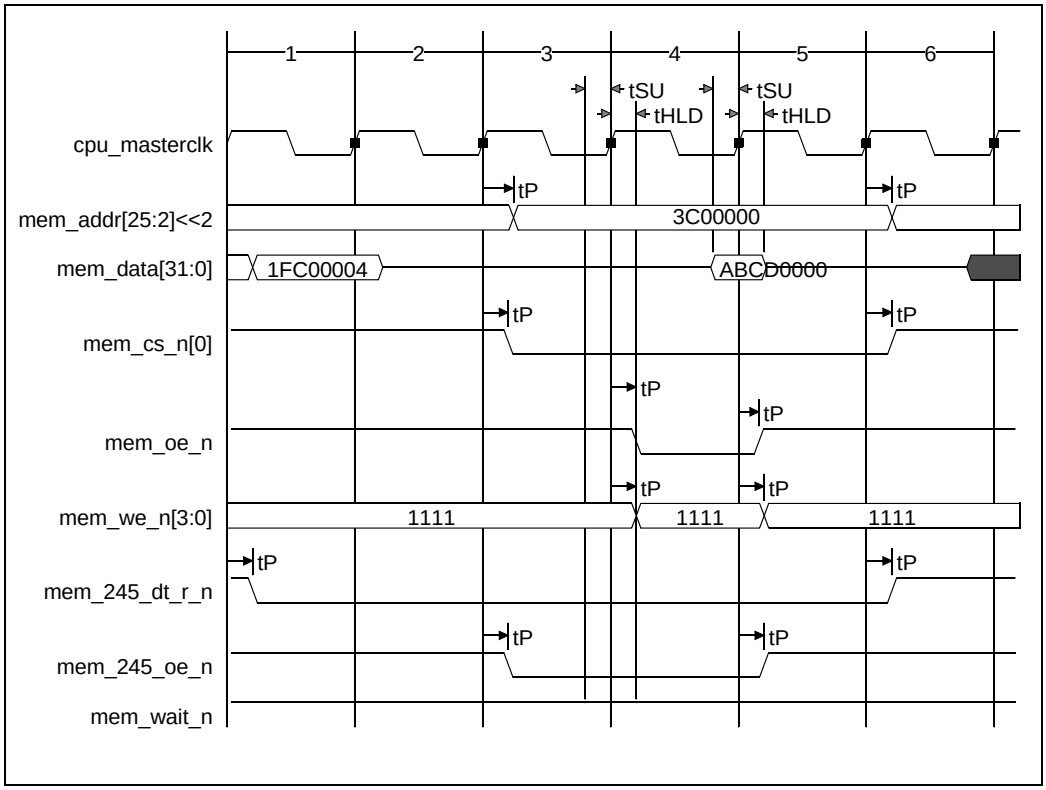


Figure 10.13 IOI 1 Word Single Read

Notes

Figure 10.14 shows an IOI-type single word memory read with 2 internally generated wait-states. This case provides 1 more wait-state beyond the required minimum of 1 wait-state. (Type=01, 245=1, WP=0, PW=10, WWS=3. RWS=2).

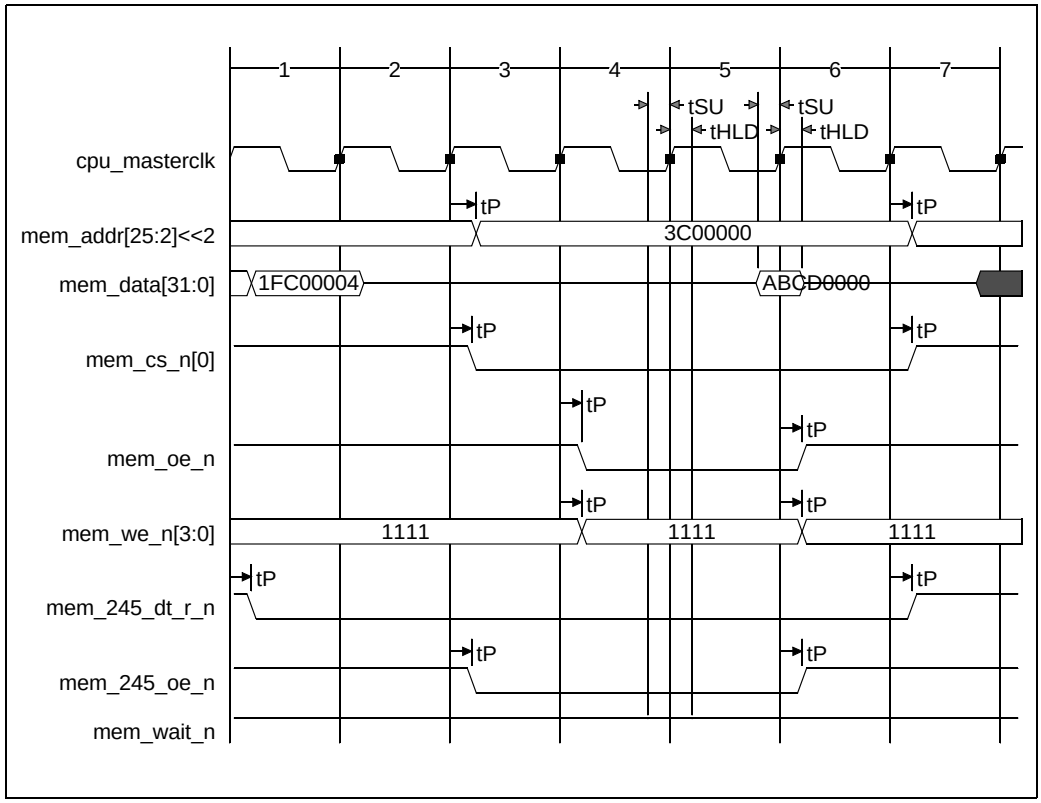


Figure 10.14 IOI 1 Word Single Read with Wait-State

Notes

Figure 10.15 shows an IOI-type single word memory write with 2 internally generated wait-states. Note that the write enable bus, mem_we_n[3:0], provides the write data strobe.

Note that additional internally generated wait-states will repeat state 4, so that mem_oe_n is continuously asserted during internal wait states.(Type=01, 245=1, WP=0, PW=10, WWS=2, RWS=1).

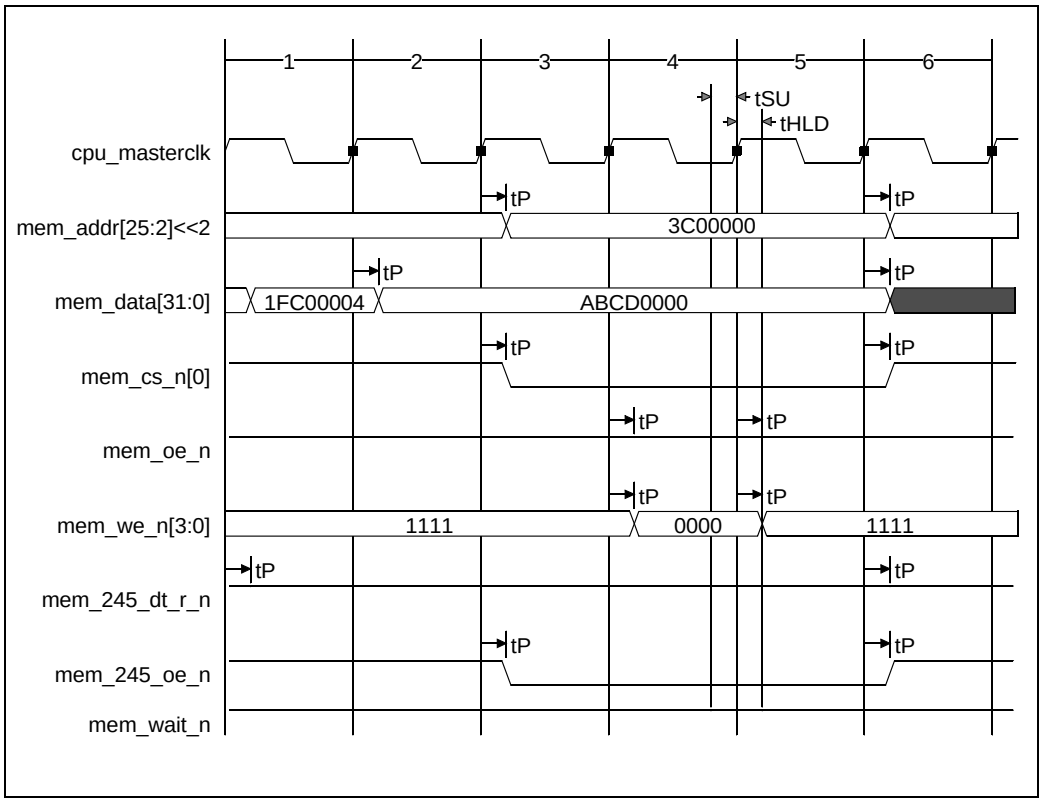


Figure 10.15 IOI 1 Word Single Write

Notes

Figure 10.16 shows an IOI-type single word memory write with 3 internally generated wait-states. This case provides 1 more wait-state beyond the required minimum of 2 wait-states. (Type=01, 245=1, WP=0, PW=10, WWS=3. RWS=2).

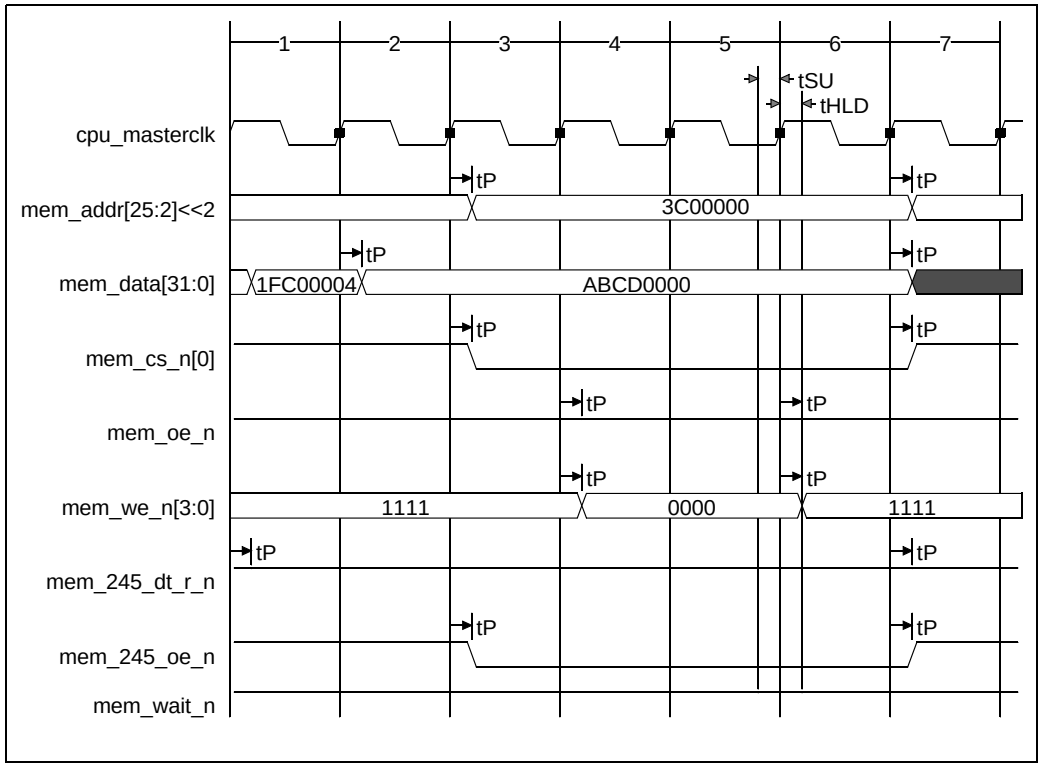


Figure 10.16 IOI 1 Word Single Write with Wait-State

Figure 10.17 shows an IOI-type four word memory burst read with 1 internally generated wait-state for each datum. Note that the read output enable, mem_oe_n, provides the read data strobe. The burst IOM-type access is not conventionally used by I/O peripherals. (Type=01, 245=1, WP=0, PW=10, WWS=2, RWS=1).

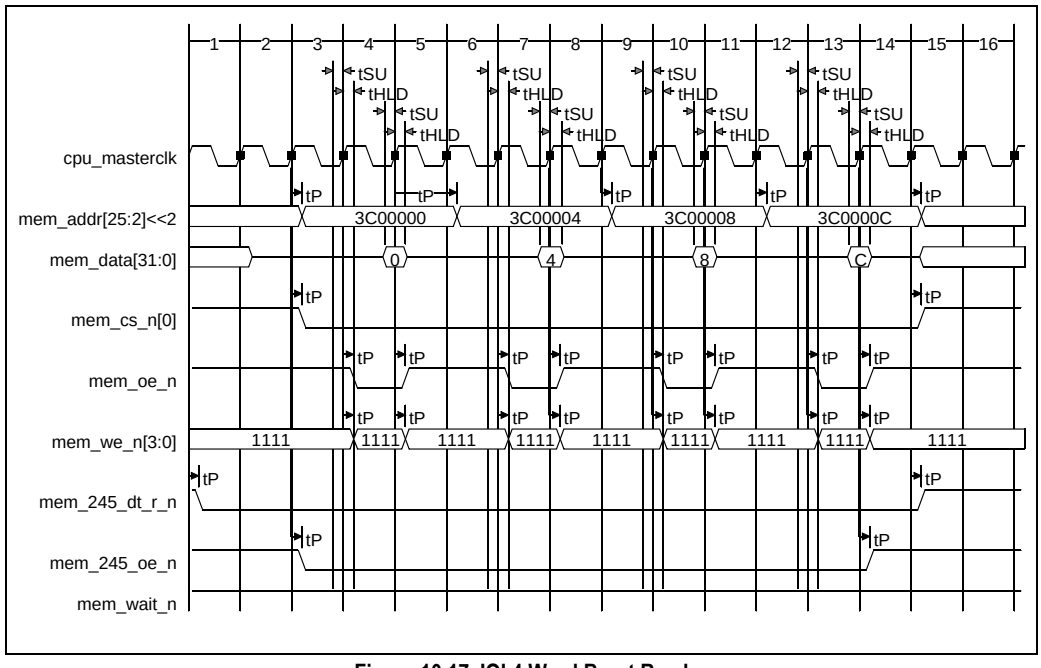


Figure 10.17 IOI 4 Word Burst Read

Notes

Figure 10.18 shows an IOI-type four word memory burst write with 2 internally generated wait-states for each data. Note that the write enable bus, mem_we_n[3:0], provides the write data strobe. The burst IOM-type access is not conventionally used by I/O peripherals. (Type=01, 245=1, WP=0, PW=10, WWS=2, RWS=1).

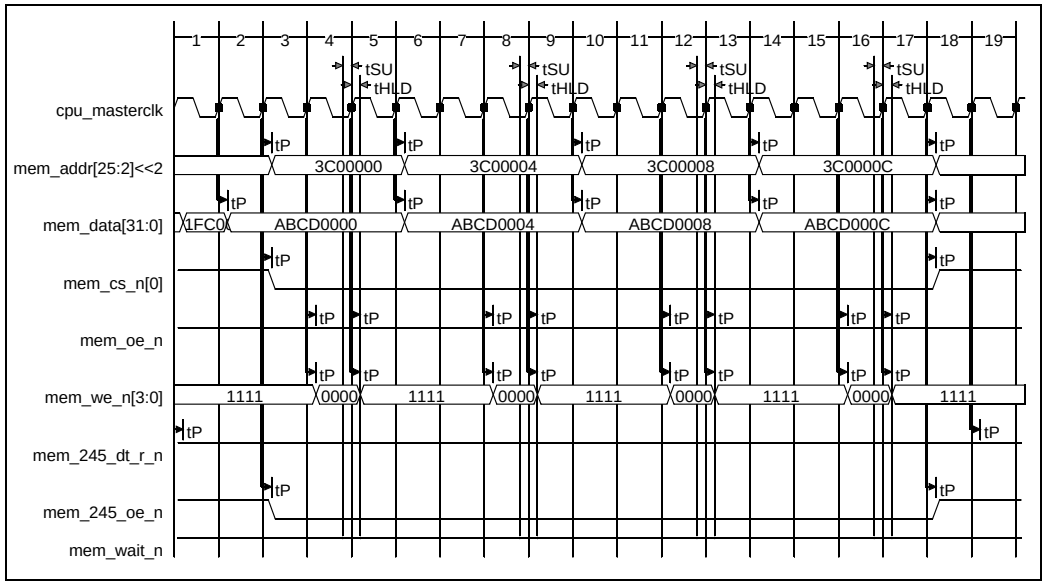


Figure 10.18 IOI 4 Word Burst Write

Figure 10.19 shows an IOM-type single word memory read with 1 internally generated wait-state. Note that the chip select, mem_cs_n[0], provides the data strobe while the output enable—or the write enables—indicate the read/write status. Also note that additional internally generated wait-states will repeat state 4, so that mem_cs_n[0] is continuously asserted during internal wait-states. (Type=10, 245=1, WP=0, PW=10, WWS=2, RWS=1).

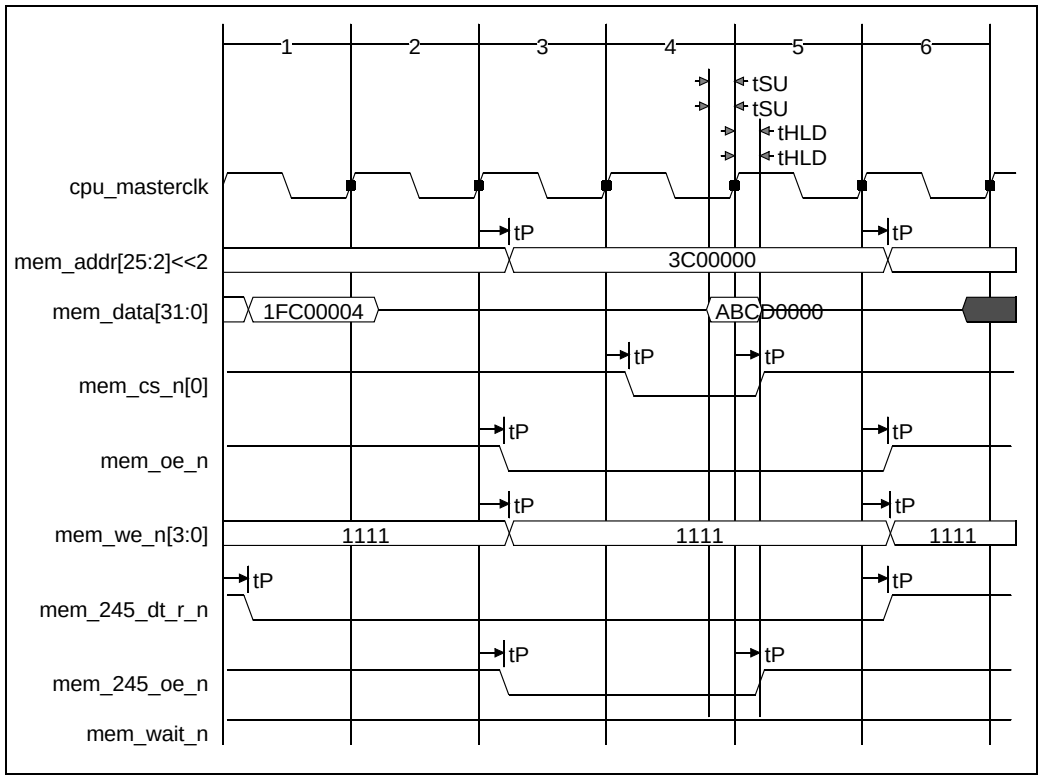


Figure 10.19 IOM 1 Word Single Read

Notes

Figure 10.20 shows an IOM-type single word memory read with 2 internally generated wait-states. This case provides 1 more wait-state beyond the required minimum of 1 wait-state. (Type=10, 245=1, WP=0, PW=10, WWS=3. RWS=2).

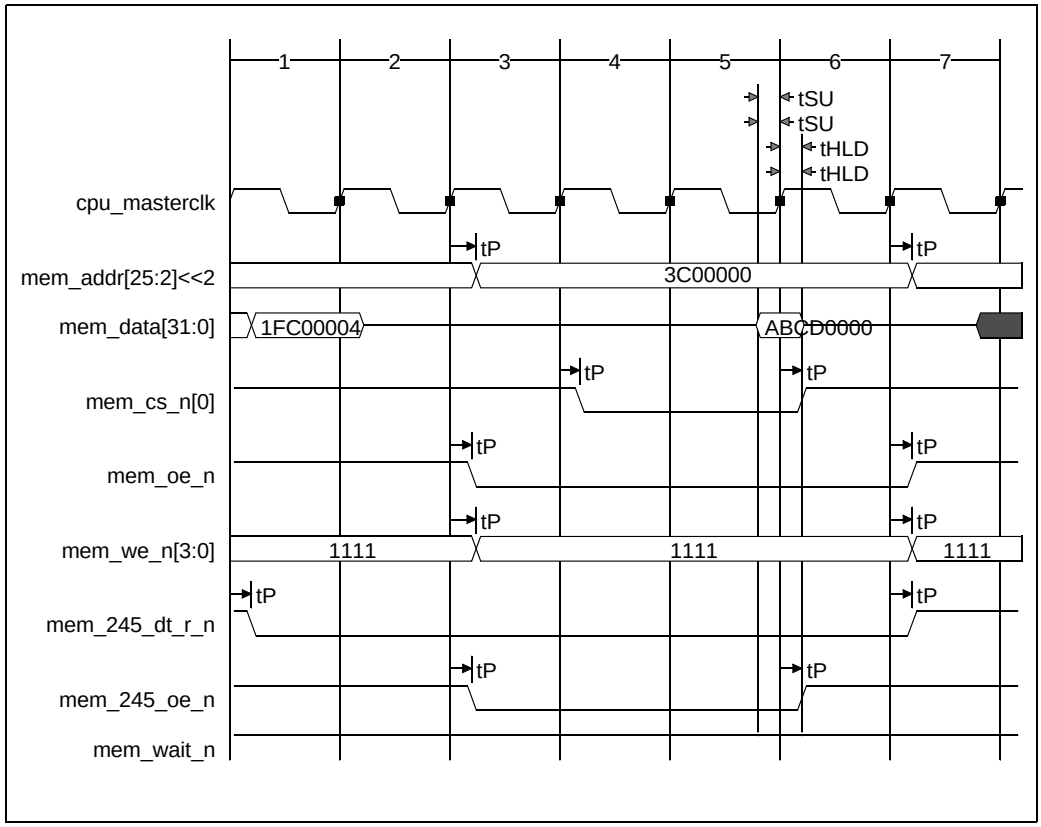


Figure 10.20 IOM 1 Word Single Read with Wait-State

Notes

Figure 10.21 shows an IOM-type single word memory write with 2 internally generated wait-states. Note that the chip select, mem_cs_n[0], provides the data strobe while the output enable, or the write enables, indicate the read/write status. Also note that additional internally generated wait-states will repeat state 4, so that mem_cs_n[0] is continuously asserted during internal wait-states. (Type=10, 245=1, WP=0, PW=10, WWS=2, RWS=1).

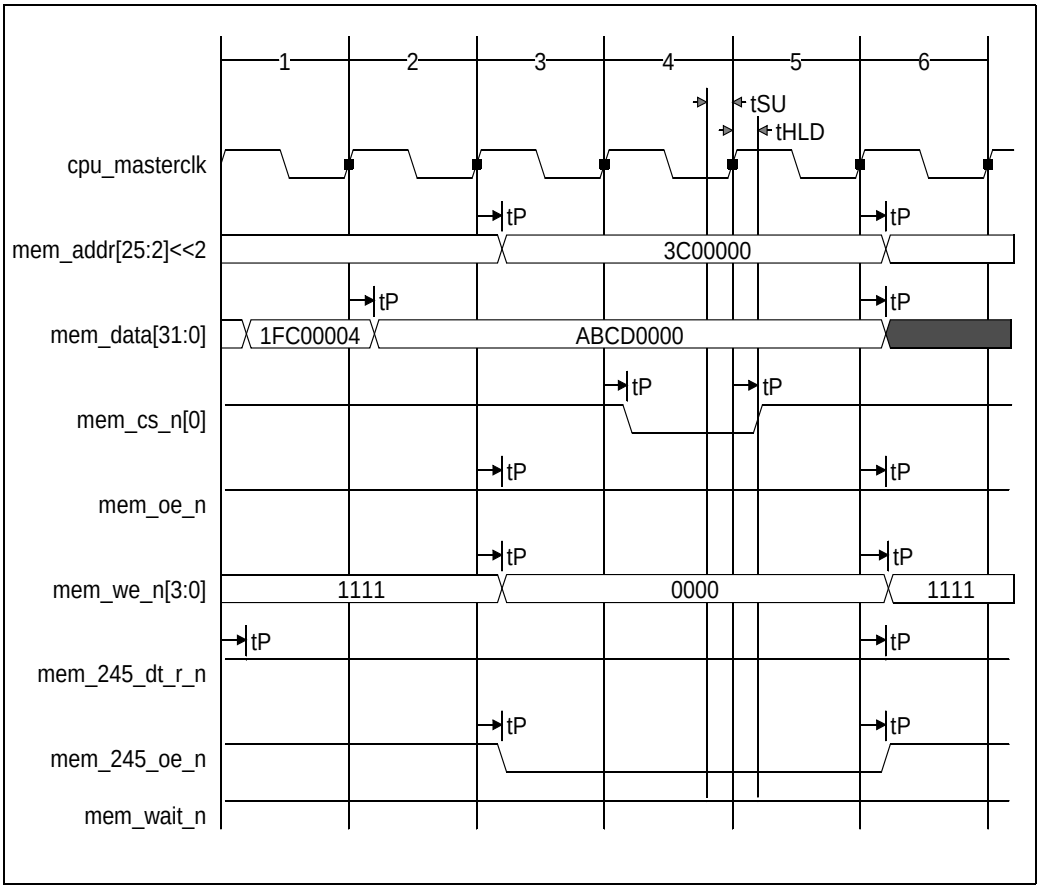


Figure 10.21 IOM 1 Word Single Write

Notes

Figure 10.22 shows an IOM-type single word memory write with 3 internally generated wait-states. This case provides 1 more wait-state beyond the required minimum of 2 wait-states. (Type=10, 245=1, WP=0, PW=10, WWS=3. RWS=2).

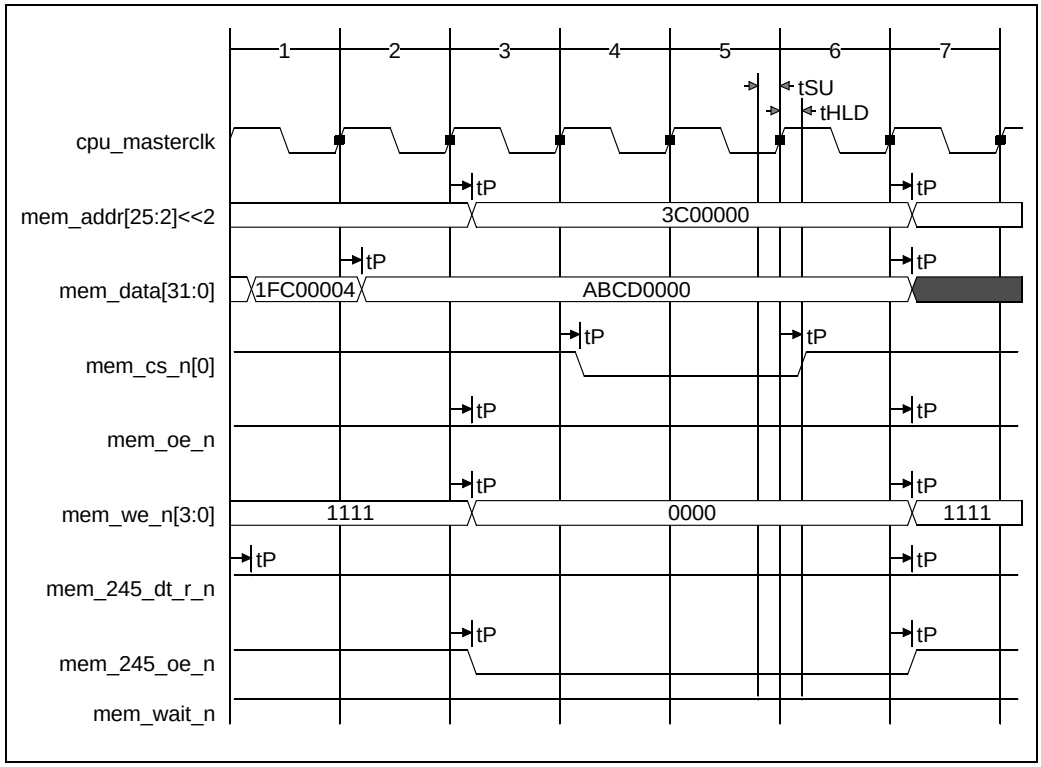


Figure 10.22 IOM 1 Word Single Write with Wait-State

Figure 10.23 shows an IOM-type four word memory read with 1 internally generated wait-state for each data. Note that the chip select, mem_cs_n[0], provides the data strobe while the output enable, or the write enables, indicate the read/write status. The burst IOM-type access is not conventionally used by I/O peripherals. (Type=10, 245=1, WP=0, PW=10, WWS=2, RWS=1).

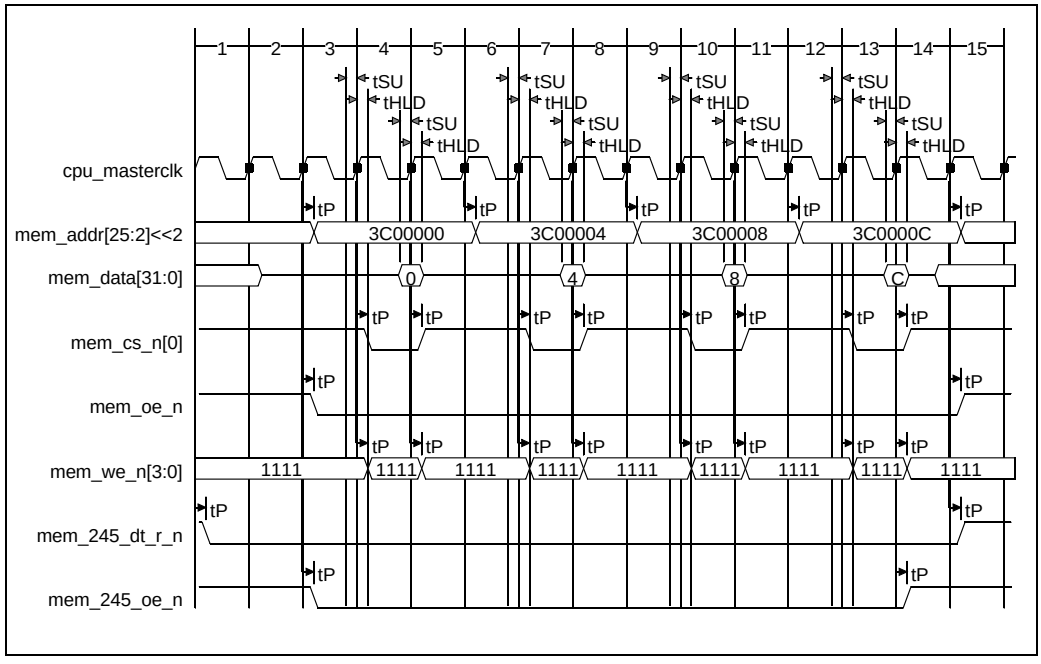


Figure 10.23 IOM 4 Word Burst Read

Notes

Figure 10.24 shows an IOM-type four word memory write with 2 internally generated wait-states for each datum. Note that the chip select, mem_cs_n[0], provides the data strobe while the output enable, or the write enables, indicate the read/write status. The burst IOM-type access is not conventionally used by I/O peripherals. (Type=10, 245=1, WP=0, PW=10, WWS=2, RWS=1).

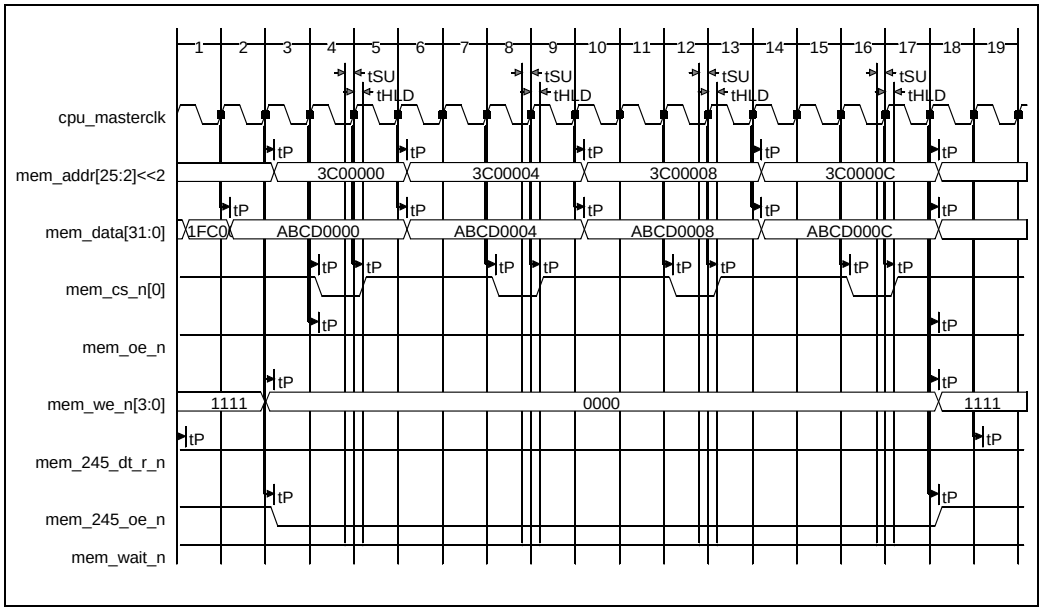


Figure 10.24 IOM 4 Word Burst Write

Figure 10.25 shows a DPM-type single word memory read with 1 internally generated wait-state. Note that both the chip select, mem_cs_n[0] and the output enable, mem_oe_n, primarily determine the read access time for the data from the SRAM.

Note that additional internally generated wait-states will repeat state 4. (Type=11, 245=1, WP=0, PW=10, WWS=2, RWS=1).

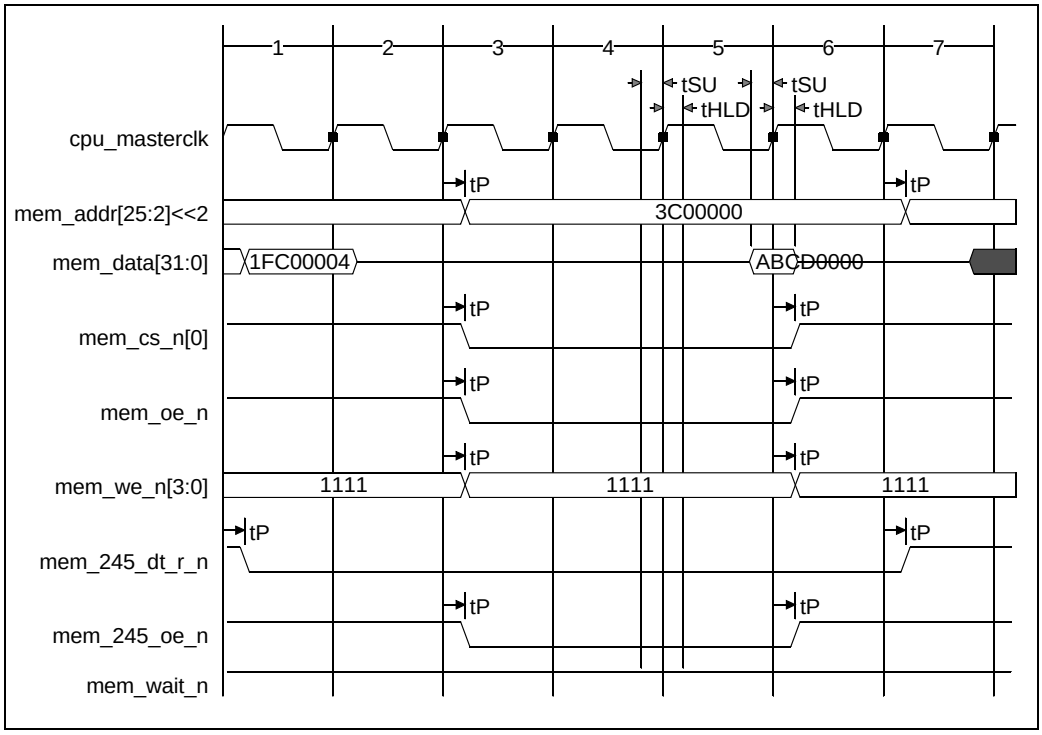


Figure 10.25 Dual-Port 1 Word Single Read

Notes

Figure 10.26 shows a DPM-type single word memory read with 1 internally generated wait-state, and then 1 externally generated wait-state is indicated by mem_wait_n asserting. Note that the internal wait-state counter starts over each time mem_wait_n is asserted, such that when mem_wait_n de-asserts the internal wait-state counter goes through a complete count before the transaction ends. (Type=11, 245=1, WP=0, PW=10, WWS=2, RWS=1).

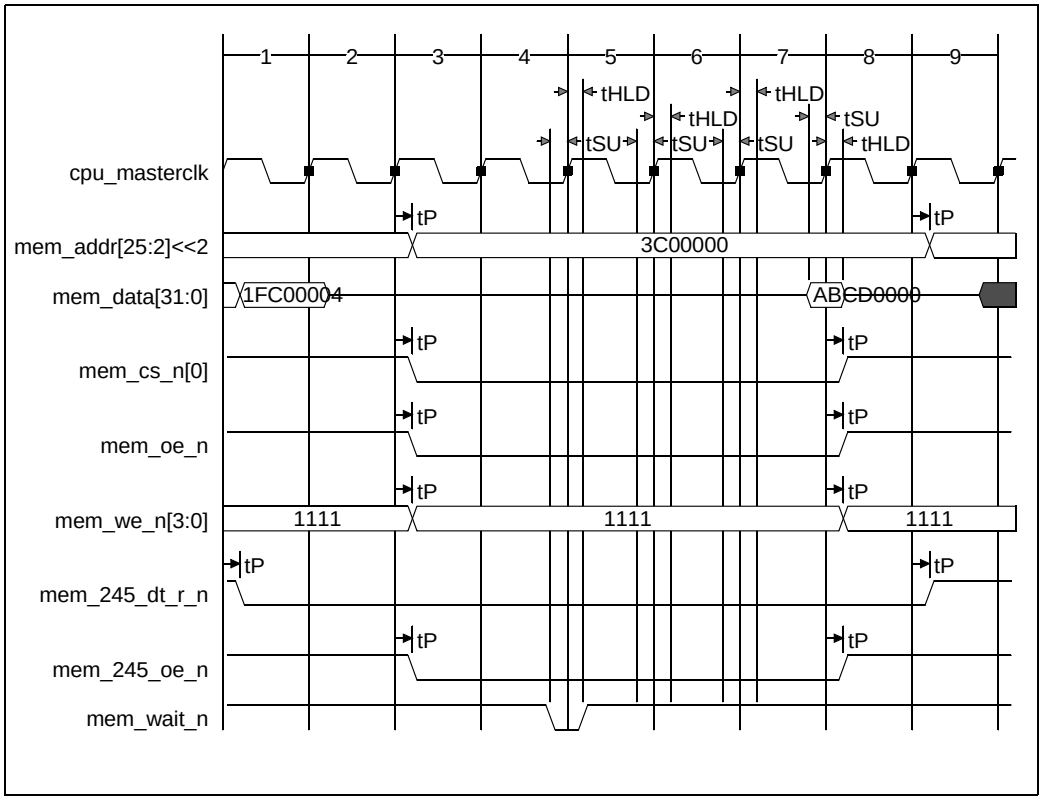


Figure 10.26 Dual-Port 1 Word Single Read with Wait-State

Notes

Figure 10.27 shows a DPM-type single word memory write with 2 internally generated wait-states. Note that the writes enables, mem_we_n[3:0], are used as the primary write strobes. Note that additional internally generated wait-states will repeat state 4. (Type=11, 245=1, WP=0, PW=10, WWS=2, RWS=1).

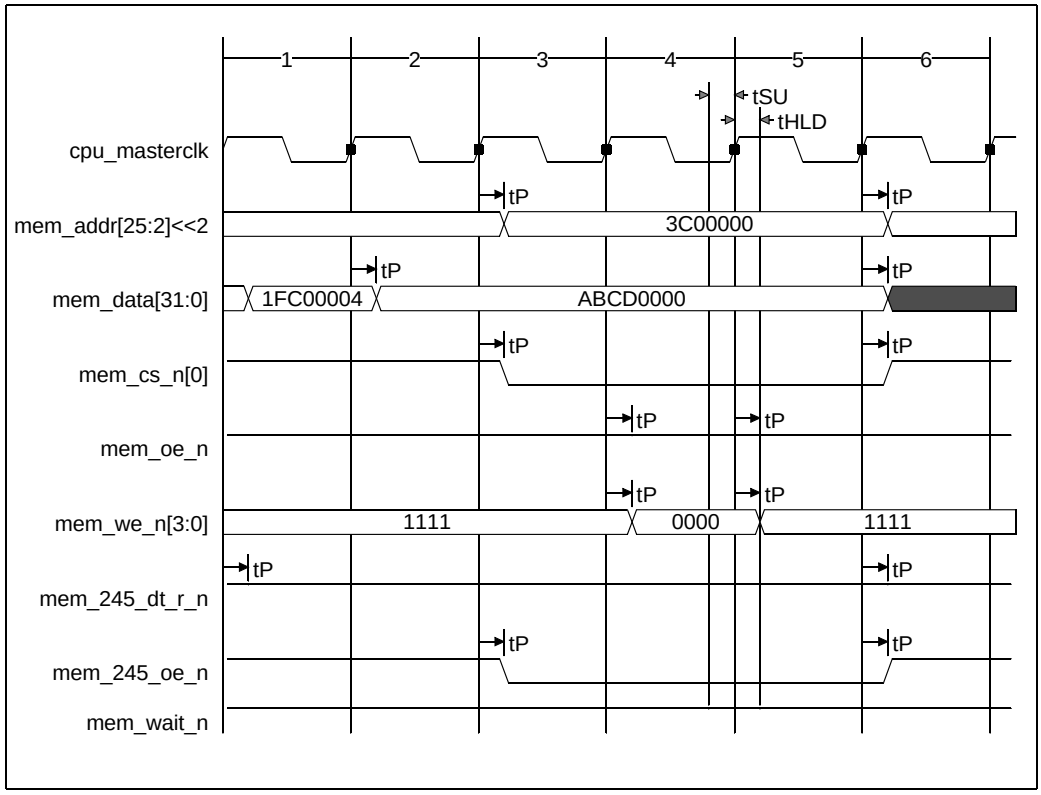


Figure 10.27 Dual-Port 1 Word Single Write

Notes

Figure 10.28 shows an SRAM-type single word memory write with 3 internally generated wait-states and then 1 externally generated wait-state as indicated by mem_wait_n asserting. This case provides 1 more wait-state beyond the required minimum of 2 wait-states. Note that if the memory controller were programmed such that 3 or more internally generated wait-states occurred, mem_wait_n is ignored until the final wait-state occurs where debug_cpu_ack_n would assert. (Type=00, 245=1, WP=0, PW=10, WWS=3, RWS=2).

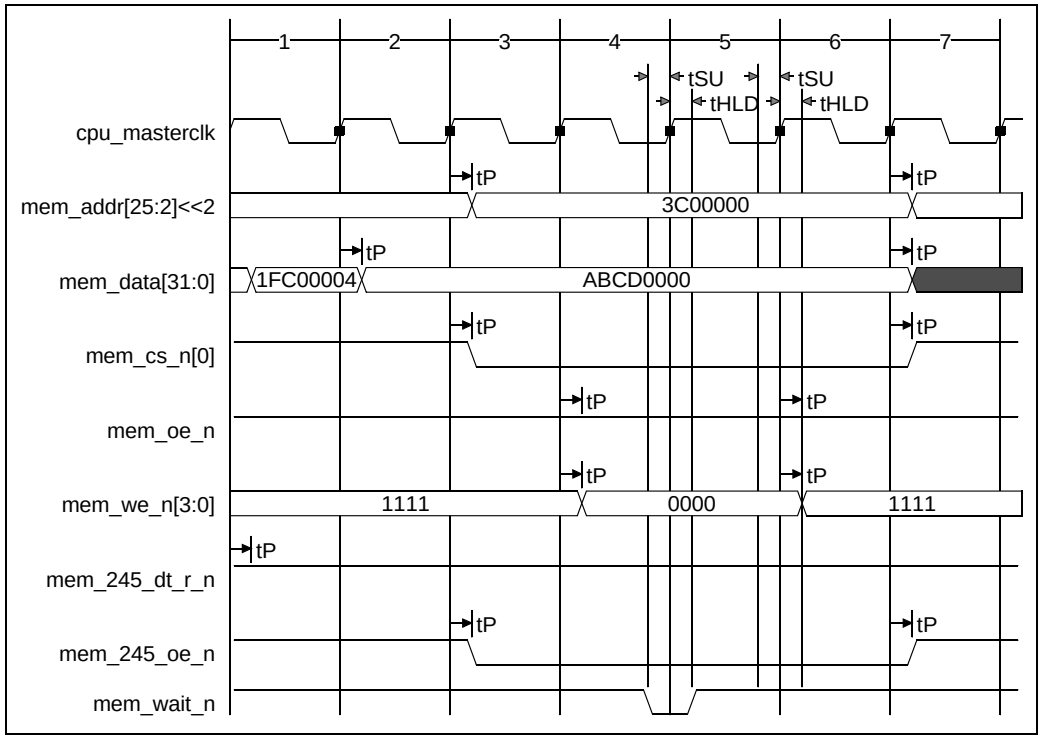


Figure 10.28 Single Word SRAM Write Transaction with Wait-State

Figure 10.29 shows a DPM-type four word memory burst read with 1 internally generated wait-state for each datum. Note that both the chip select, mem_cs_n[0] and the output enable, mem_oe_n primarily determine the read access time for the data from the SRAM. (Type=11, 245=1, WP=0, PW=10, WWS=2, RWS=1).

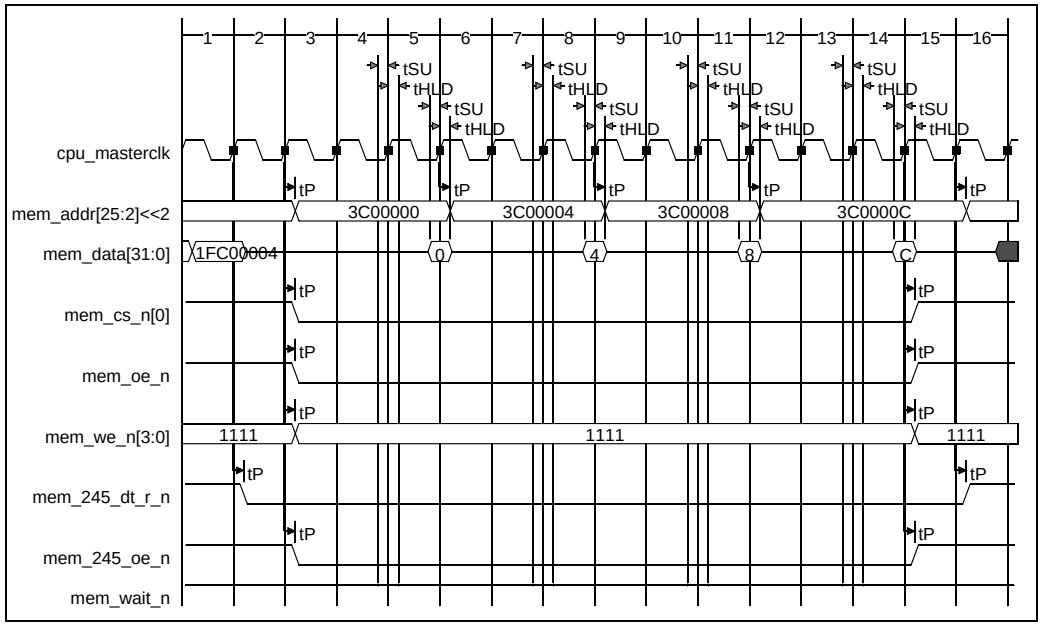


Figure 10.29 Dual-Port 4 Word Burst Read

Notes

Figure 10.30 shows a DPM-type four word memory burst write with 2 internally generated wait-states. Note that the writes enables, mem_we_n[3:0], are used as the primary write strobes. (Type=11, 245=1, WP=0, PW=10, WWS=2, RWS=1).

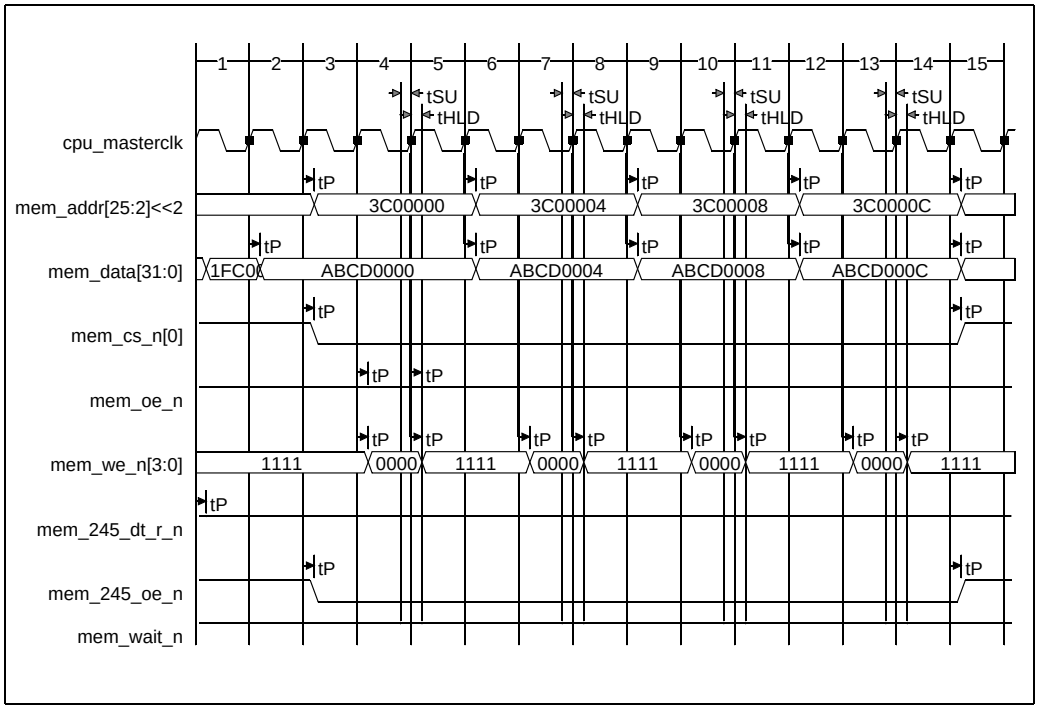


Figure 10.30 Dual-Port 4 Word Burst Write

Notes

Synchronous DRAM Controller

Notes

Introduction

The SDRAM controller supports 4 channels of 32-bit physical banks. Because each SDRAM chip/channel internally provides 2 to 4 bank arrays of memory, the 4 physical channels have a total of 8 to 16 virtual/conventional page banks. In systems using DIMMs, the 4 chip selects correspond to 2 DIMM cards. Only same size banks are supported. A total of 512 MB of SDRAM memory can be used. Each of the DRAM channels has software selectability as to how much memory space a channel uses.

Features

- *SDRAM controller (32-bit memory only)*
  - *4 banks, non-interleaved, 512 MB total (interleaving is not supported)*
  - *Automatic refresh generation in the background*
- *Software programmable options support 1 (33MHz), 2 (66MHz), or 3 clock CAS latency for 75MHz parts*
- *Software programmable Pre-charge Time and Refresh Time*
- *Supports SDRAM DIMMS or SODIMMS*

SDRAM Enhancements in Y Silicon Revision

The SDRAM memory controller is one of the modules that has been enhanced in the Y revision of the silicon. Table 11.1 outlines some of the significant differences between the Z and Y revisions. For more information on the differences between the silicon revisions, refer to Application Note AN-350, RC32334/RC32332 Differences Between Z and Y Revisions, and to the RC32334/RC32332 Device Errata, both posted on IDT's web site at www.idt.com.

Function	Z Revision	Y Revision	Comments
SDRAM address line 16	Not supported	Added	Allows the Y revision to address 256 and 512Mb SDRAM devices. Note: Use of 256Mb SDRAMs was implemented for Z revision using mem_addr[25:24], an approach that restricted their use to RC32334-based implementations.
Supported SDRAM Memory configurations	16Mb-256Mbit	16M-512Mbit. Additional configurations include64Mb: 2Mbx32128Mb: 32Mbx4	Allows the part to support larger bursts from external PCI bus masters transferring information to/from SDRAM. Note: Legacy bank address to mem_addr line mappings have changed between silicon revisions.

Table 11.1 SDRAM Differences Between Z and Y Revisions (Part 1 of 2)

Notes

Function	Z Revision	Y Revision	Comments
tWR Timing	Uses tRP to define minimum recovery period after a write.	Additional option to set recovery period to two clock periods.	
DQM Behavior	Assertion of DQM signal used to setup external transceivers for buffered SDRAM systems. DQM asserted 2 clocks early for read accesses. Chip select signal would stay asserted for two clocks beyond the access.	Option to assert DQM only with read or write command.	Simplifies view and interpretation by logic analyzers of memory cycles.
SDRAM refresh behavior	Burst mode did not support dynamic bytes enables. Burst needed all four byte enables to be active for all words in the burst.	Byte enables are sampled dynamically for every datum.	Simplifies system design and programming of memory controller.

Table 11.1 SDRAM Differences Between Z and Y Revisions (Part 2 of 2)

To ensure backwards compatibility with Z revision, the new functionality in Y revision is enabled using bits in a new register that was not included in the Z revision. Specifically, the Control Register present in the Z revision has been renamed to Primary Control Register and a new register, Secondary Control Register, has been added. Both registers are shown in Table 11.2 below.

Base Address	Register	Offset Address	Effective Address
1800_0000	SDRAM Primary Control Register	300	Base + Offset
	SDRAM Secondary Control Register	304	

Table 11.2 Modified and New SDRAM Control Registers

Notes

Block Diagram

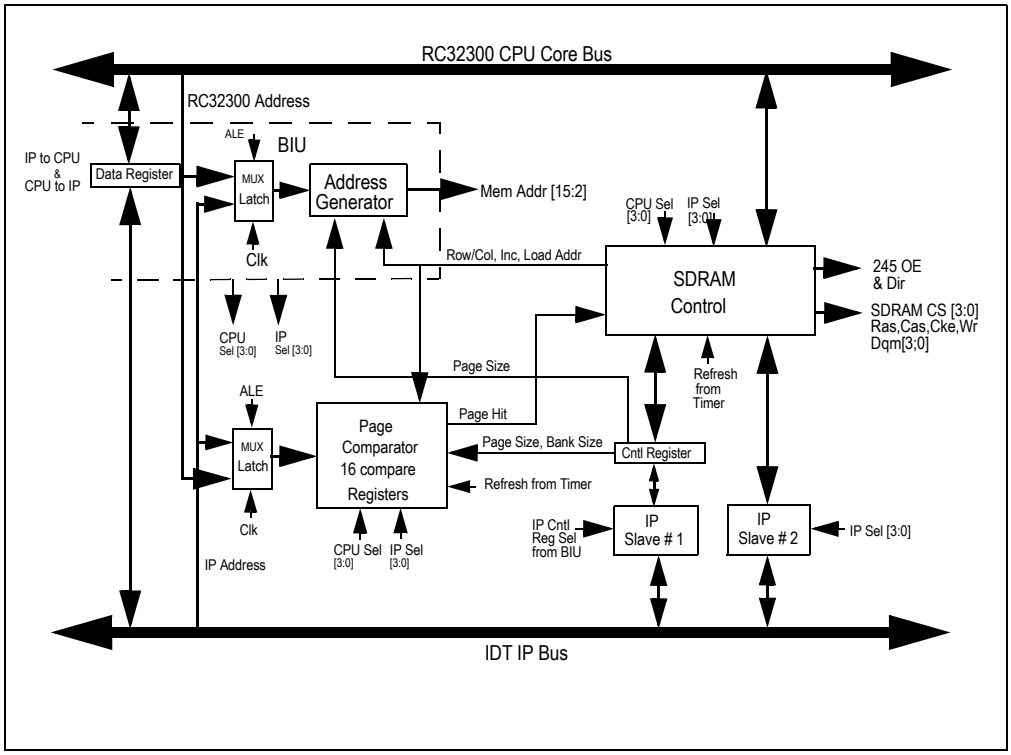


Figure 11.1 SDRAM Block Diagram

Functional Overview

The SDRAM controller provides a glueless interface to industry standard SDRAMs as well as four chip selects (sdram_cs_n[3:0]), each supporting either two or four SDRAM banks. Two banks are supported when 16 M-bit SDRAMs are used; four banks when 64 M-bit SDRAMs are used. Each SDRAM bank must have a 32-bit data path. As shown in Table 11.3, the SDRAM controller supports a wide variety of SDRAMs, allowing the 32-bit data path to be constructed using x4, x8, x16, or x32 SDRAMs.

SDRAM Size	SDRAM Organization	SDRAM Chip Select Total Memory
16 M-bit	2Mb x 4 x 2 banks	16 MB
	1Mb x 8 x 2 banks	8 MB
	512 Kb x 16 x 2 banks	4 MB
64 M-bit	4 Mb x 4 x 4 banks	64 MB
	2 Mb x 8 x 4 banks	32 MB
	1 Mb x 16 x 4 banks	16 MB
	0.5 Mb x 32 x 4 banks	8 MB
128 M-bit	8 Mb x 4 x 4 banks	128 MB (Limit 2 chip selects)[1]
	4 Mb x 8 x 4 banks	64 MB
	2 Mb x 16 x 4 banks	32 MB
	1 Mb x 32 x 4 banks	16 MB

Table 11.3 Supported SDRAMs (Part 1 of 2)

Notes

SDRAM Size	SDRAM Organization	SDRAM Chip Select Total Memory
256 M-bit (RC32334 only)	16 Mb x 4 x 4 banks	256 MB[2] (Limit 1 chip select)
	8 Mb x 8 x 4 banks	128 MB[2] (Limit 2 chip selects)
	4 Mb x 16 x 4 banks	64 MB
	2 Mb x 32 x 4 banks	32 MB
512 M-bit (RC32334 only)	16 Mb x 8 x 4 banks	256 MB[2] (Limit 1 chip select)
	8 Mb x 16 x 4 banks	128 MB[2] (Limit 2 chip selects)

Table 11.3 Supported SDRAMs (Part 2 of 2)

1. The allocated physical memory map for SDRAM is 256 MB maximum when using the User Mode. The Kernel Mode can address additional memory above 0xC000_0000, but it is generally not recommended since the User Mode cannot access this physical address space. Thus, in practice, several of the 128 M-bit, 256 M-bit, and 512 M-bit systems are limited to using only 1 or 2 of the available 4 chips selects.

The master input clock to RC32334 is cpu_masterclk, which is used as the system clock for SDRAMs. All SDRAM transactions on the memory and peripheral bus are synchronous to this clock. During SDRAM transactions, the address bus is multiplexed as shown in Table 11.4. The exact address multiplexing is dependent upon the configuration of the page size field in the SDRAM control register.

The SDRAM controller contains a single control register, since SDRAMs connected to all four chip selects must share a common configuration. The SDRAM controller does not support the burst addressing mode of SDRAMs. Instead, SDRAMs must be configured to use the pipeline command mode, allowing the SDRAM controller to simulate burst operations by issuing a new address on each clock cycle. This allows the SDRAM controller to perform linear burst operations, as required by the DMA controller, as well as supporting Subblock Address Ordering read operations as required by cache refills.

The SDRAM controller provides the control signals necessary to control two sets of external buffers, such as 74FCT245s, on the RC32334 system data bus (mem_data[31:0]). The buffer output enable (sdram_245_oe_n) pins are the enables for such buffers, while the external buffer direction (sdram_245_dt_r_n) pin controls the direction.

Note: The Memory and Peripheral Address Bus sdram_addr [13:2], corresponds to the SDRAM chip pins A11:A0.

SDRAM[16] is added by multiplexing its functionality onto mem_addr[16]. The drive strength for mem_addr[16] must be increased to SDRAM high drive strength. SDRAM[16] outputs a27, a26, a25, and a24 are based on the SDRAM RAS Mux Control field setting in the SDRAM Control register.

The RC32334 includes a dedicated SDRAM address signal, denoted sdram_addr_12 (A10), which allows transparent refreshes to assert the PRECHARGE_ALL command during SDRAM accesses, as well as the appropriate row address during the row address command. This signal should be connected to the A10 pin on the SDRAM devices.

SDRAM Organization[1]	Cycle	Memory and Peripheral Bus Address (sdram_addr[16:2])[2]														
		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
2 Mb x 4 x 2 banks 16 Mb (10-bit page)	Pin#				BA[3]	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row				a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12
	Column				a23	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
1 Mb x 8 x 2 banks 16 Mb (9-bit page)	Pin#				BA[3]	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row				a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11
	Column				a22	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2

Table 11.4 SDRAM Address Multiplexing (Part 1 of 3)

Notes

SDRAM Organization[1]	Cycle	Memory and Peripheral Bus Address (sdram_addr[16:2])[2]														
		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
512 Kb x 16 x 2 banks 16 Mb (8-bit page)	Pin#				BA[3]	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row				a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10
	Column				a21	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
4 Mb x 4 x 4 banks 64 Mb (10-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12
	Column		a25	a24	a23	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
2 Mb x 8 x 4 banks 64 Mb (9-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11
	Column		a24	a23	a22	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
1 Mb x 16 x 4 banks 64 Mb (8-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10
	Column		a23	a22	a21	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
512 Kb x 32 x 4 banks 64 Mb (8-bit page)	Pin#			BA1	BA0	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row			a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10
	Column			a22	a21	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
8 Mb x 4 x 4 banks 128 Mb (11-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a26	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13
	Column		a26	a25	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
4 Mb x 8 x 4 banks 128 Mb (10-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12
	Column		a25	a24	a23	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
2 Mb x 16 x 4 banks 128 Mb (9-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11
	Column		a24	a23	a22	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
1 Mb x 32 x 4 banks 128 Mb (8-bit page)	Pin#		BA1	BA0	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row		a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10
	Column		a23	a22	a21	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
16 Mb x 4 x 4 banks 256 Mb (11-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a27	a26	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13
	Column	a27	a26	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
8 Mb x 8 x 4 banks 256 Mb (10-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a26	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12
	Column	a26	a25	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
4 Mb x16 x 4 banks 256 Mb (9-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11
	Column	a25	a24	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2

Table 11.4 SDRAM Address Multiplexing (Part 2 of 3)

Notes

SDRAM Organization[1]	Cycle	Memory and Peripheral Bus Address (sdram_addr[16:2])[2]														
		16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
2 Mb x 32 x 4 banks 256 Mb (8-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12	a11	a10
	Column	a24	a23	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
16 Mb x 8 x 4 banks 512 Mb (11-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a27	a26	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13
	Column	a27	a26	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2
8 Mb x 16 x 4 banks 512 Mb (10-bit page)	Pin#	BA1	BA0	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
	Row	a26	a25	a24	a23	a22	a21	a20	a19	a18	a17	a16	a15	a14	a13	a12
	Column	a26	a25	a13	a12	AP[4]	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2

Table 11.4 SDRAM Address Multiplexing (Part 3 of 3)

1. a25 - a2 denote bits from the RC32334 internal physical address bus.

2. sdram_addr[14:2] pins correspond to the A12:A0 pins on SDRAM devices.

3. For 16 M-bit bank size, sdram_addr[14] duplicates sdram_addr[13] for DIMM expandability, such that the SDRAM BA pin(s) can be connected to sdram_addr[14] instead of sdram_addr[13].

4. AP - Auto Precharge (automatically precharged at end of transaction).

The sdram_ras_n, sdram_cas_n, sdram_we_n, and sdram_addr[12] signals, summarized in Table 11.5, encode the command issued to an SDRAM.

Command	Description	sdram_ras_n	sdram_cas_n	sdram_we_n	sdram_addr[12]
NOP1	No operation	H	H	H	X[1]
ACTIVE	Select active bank and row	L	H	H	Row
READ	Select bank and column, perform read (auto pre-charge disabled)	H	L	H	L
READ	Select bank and column, perform read (auto pre-charge enabled)	H	L	H	H
WRITE	Select bank and column, perform write (auto pre-charge disabled)	H	L	L	L
WRITE	Select bank and column, perform write (auto pre-charge enabled)	H	L	L	L
REFRESH	Enter self refresh mode	L	L	H	X[1]
PRECHARGE	Deactivate row in bank (selected by BA)	L	H	L	L
PRECHARGE	Deactivate row in all banks	L	H	L	H

Table 11.5 SDRAM Command Encoding

1. X = Don't care.

Notes

Base Address Decoding

The SDRAM base (SDRAM[3|2|1|0]BASE) and SDRAM mask (SDRAM[3|2|1|0]MASK) registers[1] control the address decoding for each SDRAM chip select. The SDRAM mask register bit settings control the bits used for address decoding. When a bit in the SDRAM mask register equals one, the corresponding address bit is active in address comparisons.

If a bit in the SDRAM mask register equals zero, then the corresponding address bit does not participate in address comparisons. All active address bits not masked by the SDRAM mask register are compared to the value in the SDRAM base register. If they all match, then the corresponding SDRAM chip select is asserted. The software selectable regions are as listed in Table 11.6.

Default Physical Address Range		Default Base Setting	Default Mask Setting	SDRAM Register Description
From	To			
0000_0000	000F_FFFF	0000_0000	FFF0_0000	Base/Mask Address Bank0
0100_0000	010F_FFFF	0010_0000	FFF0_0000	Base/Mask Address Bank1
0100_0000	010F_FFFF	0010_0000	FFF0_0000	Base/Mask Address Bank2
0100_0000	010F_FFFF	0010_0000	FFF0_0000	Base/Mask Address Bank3

Table 11.6 Base Address and Base Mask Address Map

Note: Because Bank1, Bank2, and Bank3 default to the same base register value, all three registers must be programmed by the boot software, when initializing the DRAM interface to valid Base and Mask register values before initial access.

Page Row Comparators

Bank row page address comparators are used by the SDRAM controller to speed up consecutive multiple bus transactions to the same row of an already active bank, i.e., where the upper row address bits are constant but the lower column address bits are changing. Each chip select supports up to four bank page row address comparators: two are used with 16 M-bit SDRAMs and all four are used with 64 M-bit SDRAMs. Although each bank row page address comparator is 12 bits in size (a31:a20), not all bits are used in all SDRAM configurations. The bank page row address comparators support 1024, 2048 & 4096 byte pages, and because even the smallest application has at least 2 pages to select from, the page comparators will typically be left on as defined by CAS A10 (sdram_addr[12]). To decode which page comparator is being used, use RAS A11 (sdram_addr[13]) (16M-bit) or RAS A12/A13 (sdram_addr[14][15]) (64M-bit).

When the CPU performs a read or write operation to SDRAM space, the bank page row address comparator associated with the SDRAM bank selected is checked. If the bank was left active and the value in the comparator matches the SDRAM row address, then the access can be made without first closing the currently active page and then opening a different page.

If the active page in the bank page row address comparator does not match the SDRAM row address, before the access can occur, the active page must first be closed (precharged) and the correct page made active. Finally, if there is no active page in the bank, the required page must first be made active and then the access can occur.

Burst Support

RC32334 does not use the burst addressing mode of SDRAM chips. Instead, the RC32334 uses the “pipeline command” mode, which imitates a burst by issuing a new address on each clock. The RC32334 is able to burst in subblock address order on bursts from the RC32300 CPU Core as well as burst in linear address order on bursts up to 1KB. from DMA. Figure 11.2 shows the subblock retrieval method.

1. The Bank1 range priority overrides the Bank0 range. Thus, if the physical address of a memory transaction is such that Bank1 is selected, it will override the Bank0 row select. Similarly, Bank3 overrides Bank2, which overrides Bank1.

Notes

```
Start Address[3:0] Burst Sequence
0                  (0,4,8,C)
4                  (4,0,C,8)
8                  (8,C,0,4)
C                  (C,8,4,0)
```

Figure 11.2 Subblock Ordered Retrieval Method

RAS/CAS Address MUX

Using an RAS/CAS multiplexer—where extra address bits can be used for various chip types (1Mx16, 2Mx8, 4Mx4)—provides 32-bit support. Because common SDRAM DIMMs come in a 64-bit width, some users may want to populate RC32334 boards with discrete package(s) in order to provide 32-bit wide memory.

Alternatively, users can tie a 64-bit DIMM module's data outputs and byte masks together to form two banks of 32-bit SDRAM. Systems consisting of more than eight chips should buffer the address lines. 64M-bit parts, which use 4 internal banks, are supported by using an additional address pin as the additional signal. Two additional page comparators are added.

If the SDRAM controller's control register bank size is selected to be 16M bank size, then during the Row address phase sdram_addr[13] (BA) will be duplicated on sdram_addr [14] for DIMM upward compatibility.

Refresh Timer

The SDRAM timer is enabled/disabled through software so that self-refresh can be invoked. An internal interrupt can be optionally generated when the timer overflows. In systems with no SDRAM, the SDRAM refresh timer can be used as a general purpose timer. This option is enabled by setting the sdram_enable_n bit (31) in the SDRAM control register to zero, which disables the queueing of SDRAM refresh transactions when the timer expires. The TO sticky bit in the RTC register is an input to the interrupt controller. A refresh will close all open pages.

Error Recovery

The SDRAM controller is required to abort gracefully on a bus error and bus time-out. Both bus errors and bus time-outs latch the present physical address into the BusError Address Register. This implies that the bus error controller can only assert buserror_n up until the first ack_n would be returned. Both ack_n and busreq_n are de-asserted when buserror_n is asserted, such that the user can connect retry_n instead of buserror_n if desired.

On a bus error, the memory controller must skip to the transaction end state of the primary state machine and return all outputs to their transaction end values. A bus time-out may occur at any time during the bus transaction, even after the first ack_n has been returned. If the time-out occurs, then ack_n is immediately returned and if a burst, the transaction continues.

SDRAM Initialization

Before using, SDRAMs must be powered up and initialized in a predefined manner. Each SDRAM contains a mode register which defines the specific mode of operation for the SDRAM. The mode register selects: the burst length, the burst type, CAS latency, operating mode, and write burst mode. The mode register is programmed using an SDRAM LOAD MODE REGISTER command.

To support compatibility within a wide range of devices, the SDRAM controller does not directly support the SDRAM LOAD MODE REGISTER command. Instead, this command must be synthesized using an SDRAM custom transaction, initiated as follows:

- Select one or all four of the chip selects (sdram_cs_n[3:0]) in the CS field of the SDRAM control register
- Program the sdram_addr[12] status by setting the sdram_addr[12] bit in the SDRAM control regis-

Notes

ter, which then determines the state of the SD_ADDR[12] pin during an SDRAM custom transaction[1]

- Program the Write Enable status by setting the WE bit in the SDRAM control register, which then determines the state of the DWEN pin during an SDRAM custom transaction
- Program the RAS and CAS status by setting the RAS and CAS bits in the SDRAM control register, which specifies the state of the RASN and CASN signals during an SDRAM custom transaction.

On the next decoded SDRAM memory cycle, a transaction will be issued to the SDRAM with the command programmed in the SDRAM control register. For the Load Mode Register command, the lower address field bits (A13:A0) determine the value programmed into the SDRAM mode register. The chip select signals selected in the chip select field are asserted for one clock cycle and are reset after the command has been executed.

The state of the sdram_ras_n, sdram_cas_n, and sdram_we_n signals reflects the state programmed into the SDRAM control register, until a new command value is written into the SDRAM control register. The state of the sdram_cke signal is reflected directly from the programmed state. The address bus is driven with the column address. If the processor operation was a write[2], the data bus is driven with the data to be written. Using this mechanism, most SDRAM commands, including LOAD MODE REGISTER, may be synthesized by the RC32334. After the SDRAM custom transaction completes, the value of the chip select field in the SDRAM control register is automatically reset back to zero.

Register Definitions

The Base Address and Base Mask registers allow selection of the address range to be decoded for each channel. The address map for base address, base mask, and primary and secondary control registers is provided in Table 11.7. Through the SDRAM Primary and Secondary Control registers, various SDRAM features and options are enabled, as shown in Figure 11.3 and Table 11.8 for primary and Figure 11.4 and and Table 11.9 for secondary.

Base Address	Register	Offset Address	Effective Address
1800_0000	DRAM Base Address Register 0	C0	Base + Offset
	DRAM Base Mask Register 0	C4	
	DRAM Base Address Register 1	C8	
	DRAM Base Mask Register 1	CC	
	DRAM Base Address Register 2	D0	
	DRAM Base Mask Register 2	D4	
	DRAM Base Address Register 3	D8	
	DRAM Base Mask Register 3	DC	
	SDRAM Primary Control Register	300	
	SDRAM Secondary Control Register	304	

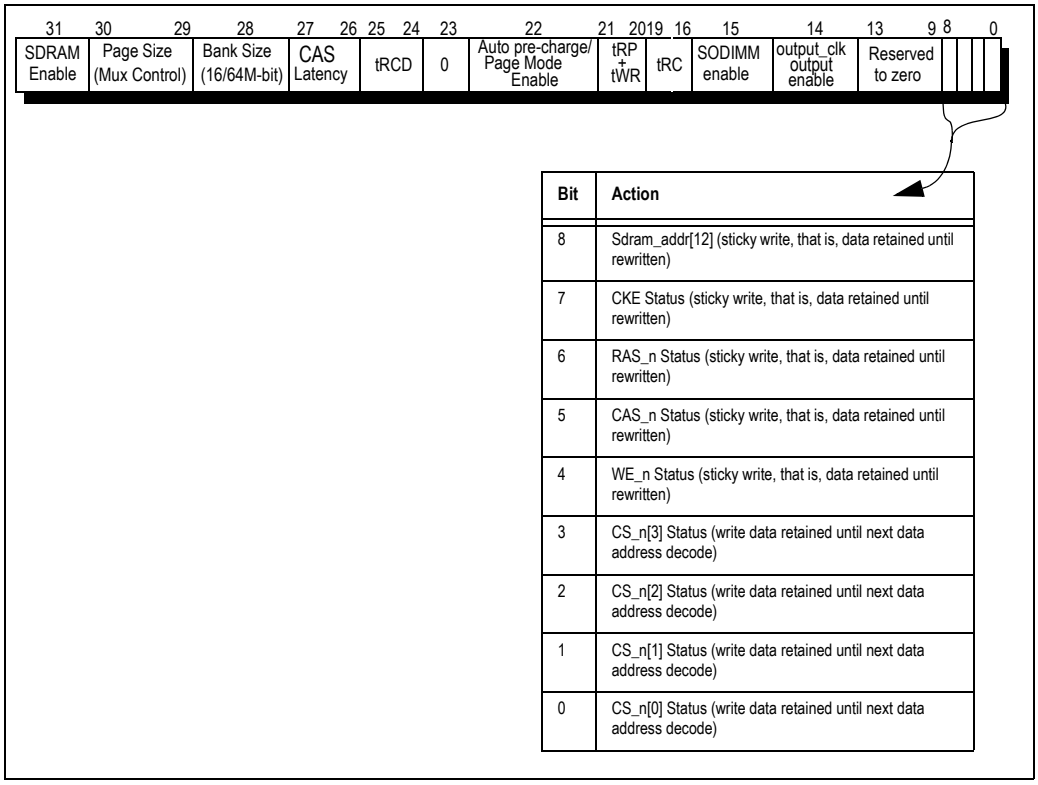
Table 11.7 SDRAM Register Address Map

1. Set this bit only for a precharge command.
2. The RC32300 CPU core performs a read or write operation to SDRAM space. This causes the RC32334 to assert the SDRAM chip selects programmed in the CS field of the control register, drive the address bus with the address of the SDRAM column address, and drive the SDRAM custom command programmed in the SDRAM register. In addition, if the CPU core performed a write operation to SDRAM space, then the data bus is driven with the data written by the CPU.

Notes

SDRAM Control Registers

SDRAM Primary Control Register



Bit	Action
8	Sdram_addr[12] (sticky write, that is, data retained until rewritten)
7	CKE Status (sticky write, that is, data retained until rewritten)
6	RAS_n Status (sticky write, that is, data retained until rewritten)
5	CAS_n Status (sticky write, that is, data retained until rewritten)
4	WE_n Status (sticky write, that is, data retained until rewritten)
3	CS_n[3] Status (write data retained until next data address decode)
2	CS_n[2] Status (write data retained until next data address decode)
1	CS_n[1] Status (write data retained until next data address decode)
0	CS_n[0] Status (write data retained until next data address decode)

Figure 11.3 SDRAM Primary Control Register Fields

Bit	Field	Description
31	SDRAM Controller Enable	Note that if the DRAM Base Address range is decoded and the SDRAM controller is not enabled, and an SDRAM access is attempted, a bus error will occur. **Value** / **Description** 1 — SDRAM Controller Enabled 0 — SDRAM Controller Disabled (default)
30:29	SDRAM RAS Mux Control	Shifts the RAS Address to MemAddr bit assignments. **Value** / **Action** 11 — No shift (11-bit CAS) 10 — Shift 1 (10-bit CAS) 4Mx4 SDRAM chips 01 — Shift 2 (9-bit CAS) 2Mx8 SDRAM chips (default) 00 — Shift 3 (8-bit CAS) 1Mx16 SDRAM chips

Table 11.8 SDRAM Primary Control Register Field Descriptions (Part 1 of 4)

Notes

Bit	Field	Description
28	SDRAM Bank Size Field A	Along with SDRAM Bank Field Size B, selects which address bits are used for the BA pin(s). **Value** / **Description** 1 — 16 M-bit (2 banks) (default) 0 — 64 M-bit (4 banks) (typical) Note: Use for x4, x8, x16 but not x32 wide 64 M-bit parts. 128 M-bit (4 banks) Note: Use for x4, x8, x16, x32 wide 128 M-bit parts.
27:26	CAS Latency (CL)	Implements a number of clocks needed for the CAS phase. The default number of clocks is two. **Value** / **Action** 11 — 3 clocks 10 — 2 clocks (Default) 01 — 1 clock 00 — Reserved A CAS latency of one is not supported by all SDRAM manufacturers.
25:24	Active to R/W Command Clocks (RCD)	Value Description **Value** / **Action** 11 — Reserved 10 — 3 clocks (Default) 01 — 2 clocks 00 — 1 clock
23	Reserved	Set to 0
22	Bank Auto Pre-charge / Bank Remains Active	This value controls sdram_addr_12 (A10) during the CAS Phase and is used to control if the SDRAM controller assumes after each single or burst access that the current bank and bank page row comparator are to be de-activated (similar to non-Page Mode DRAM); or that the current bank and bank page row comparator are to remain active in anticipation of a possible follow-on access within the same bank row (similar to Page Mode DRAM). **Value** / **Description** 1 — Bank auto pre-charge (bank row de-activated after each access) 0 — Bank remains active (bank row left-activated after each access)

Table 11.8 SDRAM Primary Control Register Field Descriptions (Part 2 of 4)

Notes

<table>
<tr><th>Bit</th><th>Field</th><th>Description</th></tr>
<tr><td>21:20</td><td>Pre-charge clocks / pre-charge to active command clocks (tRP) and Write Recovery Time Clocks (tWR) (tRP+tWR)</td><td>Number of clocks used for SDRAM pre-charge + write recovery time
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>11</td><td>4 clocks (default)</td></tr>
<tr><td>10</td><td>3 clocks</td></tr>
<tr><td>01</td><td>2 clocks (typical)</td></tr>
<tr><td>00</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>19:16</td><td>Refresh Transaction Clocks (RC)</td><td>Refresh to active clocks 1,2,3,4
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>15-12</td><td>Reserved</td></tr>
<tr><td>11-3</td><td>Number of clocks used during a refresh operation (default is 8)</td></tr>
<tr><td>2-0</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>15</td><td>SODIMM enable</td><td>Enables RC32334 controller to operate with SODIMM-144 devices. If SDRAM control register bit 15 is set to 1, then logically OR each pair of SDRAM chip selects on two new pins, and use the present chip selects for the odd chip select byte masks.
sdram_s_n[1] corresponds to chip select 3 and 2.
sdram_s_n[0] corresponds to chip select 1 and 0.
sdram_cs_n[3:0] correspond to the byte enable DQM signals for chip select 3 and 1.
sdram_bemask_n[3:0] correspond to the byte enable DQM signals for chip select 2 and 0.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>SODIMM mode enabled</td></tr>
<tr><td>0</td><td>SODIMM mode disabled</td></tr>
</table></td></tr>
<tr><td>14</td><td>Output Clk</td><td>Allows the output_clk signal to be driven as an output from the RC32334. When disabled, output_clk signal is tri-stated. Default value is 1.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Enable output_clk signal as output</td></tr>
<tr><td>0</td><td>output_clk signal output disabled</td></tr>
</table></td></tr>
<tr><td>13:9</td><td>Reserved</td><td>Reserved to zero.</td></tr>
<tr><td>8:4</td><td>sdram_addr[12], CKE,RAS_n, CAS_n,WE_n Status</td><td>The values in these register bits assert or de-assert the corresponding pins as synced with the CS_n status bits.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>High (default)</td></tr>
<tr><td>0</td><td>Low</td></tr>
</table></td></tr>
</table>

Table 11.8 SDRAM Primary Control Register Field Descriptions (Part 3 of 4)

Notes

Bit	Field	Description
3:0	CS_n[3:0]	Write low waits until the next bank address decode and, while in command write, asserts the pin low for 1 clock and de-asserts the pin high after the command write occurs. **Value** / **Description** 1 — High (default) 0 — Low

Table 11.8 SDRAM Primary Control Register Field Descriptions (Part 4 of 4)

SDRAM Secondary Control Register

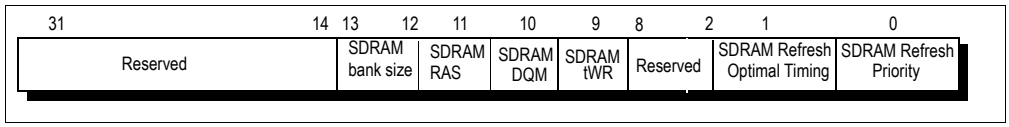


Figure 11.4 SDRAM Secondary Control Register Fields

Bit	Field	Description
31:14	Reserved	
13:12	SDRAM Bank Size Field B	Along with SDRAM Bank Size Field A, selects which address bits are used for the BA pin(s). **Value** / **Description** 11 — Reserved 10 — 256 M-bit or 512 M-bit (4 banks) 01 — 64 M-bit with x32 wide parts (4 banks) 00 — Use SDRAM Bank Size Field A (default
11	SDRAM RAS Mux Control B	Along with SDRAM RAS Mux Control Field, shifts the RAS Address to MemAddr bit assignments. **Value** / **Description** 1 — If SDRAM RAS Mux Control Field = 00, then Shift 4 (12-bit CAS). If SDRAM RAS Mux Control Field ≠ 00, then the Shift is undefined (reserved). 0 — Use the SDRAM RAS Mux Control Field (default).

Table 11.9 SDRAM Secondary Control Register Field Descriptions (Part 1 of 2)

Notes

<table>
<tr><th>Bit</th><th>Field</th><th>Description</th></tr>
<tr><td>10</td><td>SDRAM DQM</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Assert sdram_bemask_n (DQM) only with read or write command.</td></tr>
<tr><td>0</td><td>Assert sdram_bemask_n (DQM) earlier and deassert later for external transceiver setup and hold (default).</td></tr>
</table>
</td></tr>
<tr><td>9</td><td>SDRAM tWR</td><td>If the tWR field is set, the SDRAM Controller will use the tWR field rather than the tRP field to time write recovery periods. The tRP field can then be programmed to the optimal tRP period without regard to adding in the tWP period.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>Use 2 clocks for tWR independent of tRP setting (PC100+ setting).</td></tr>
<tr><td>0</td><td>Use part of tRP time to include tWR (default/ Legacy SDRAM setting).</td></tr>
</table>
Note: tWR is a timing parameter that requires a minimum recovery period after a write. The RC32334 adds an option to optimize PC-100/133 writes using the Secondary Control Register bit 9.</td></tr>
<tr><td>8:2</td><td colspan="2">Reserved</td></tr>
<tr><td>1</td><td>SDRAM Refresh with Optimal Timing</td><td>
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>New Refresh with optimal Timing behavior. Access after refresh issues an active command with no preceding precharge (typical).</td></tr>
<tr><td>0</td><td>Legacy behavior. Page mode (RAS-left-asserted) access after refresh issues a precharge (default).</td></tr>
</table>
</td></tr>
<tr><td>0</td><td>SDRAM Refresh Priority</td><td>.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>New Refresh Priority. Refresh occurs before any new read or write command (typical).</td></tr>
<tr><td>0</td><td>Legacy behavior. Refresh occurs during idle cycles after read or write commands (default).</td></tr>
</table>
</td></tr>
</table>

Table 11.9 SDRAM Secondary Control Register Field Descriptions (Part 2 of 2)

Notes

Timing Diagrams

Figure 11.5 shows an SDRAM non-page burst read, as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if the page mode is turned off. Because no precharge occurs, the row address is captured immediately and a tRCD active command to r/w command delay—in this case 2 clocks—then occurs. Finally, the column addresses are then captured.

Note that there is CAS latency from the column address until the first data appears, which in this case is 2 clocks. Also, in this case, auto precharge has been programmed—as indicated by the AP symbol—by the col3 sdram_addr. Finally, the beginning of the next transaction is shown. A minimum pre-charge time occurs, however, at least 4 clocks coincidently, prior to the next transaction because of the RC32300 CPU core BTA protocol. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=1, RP=4, RC=8, Status=FF).

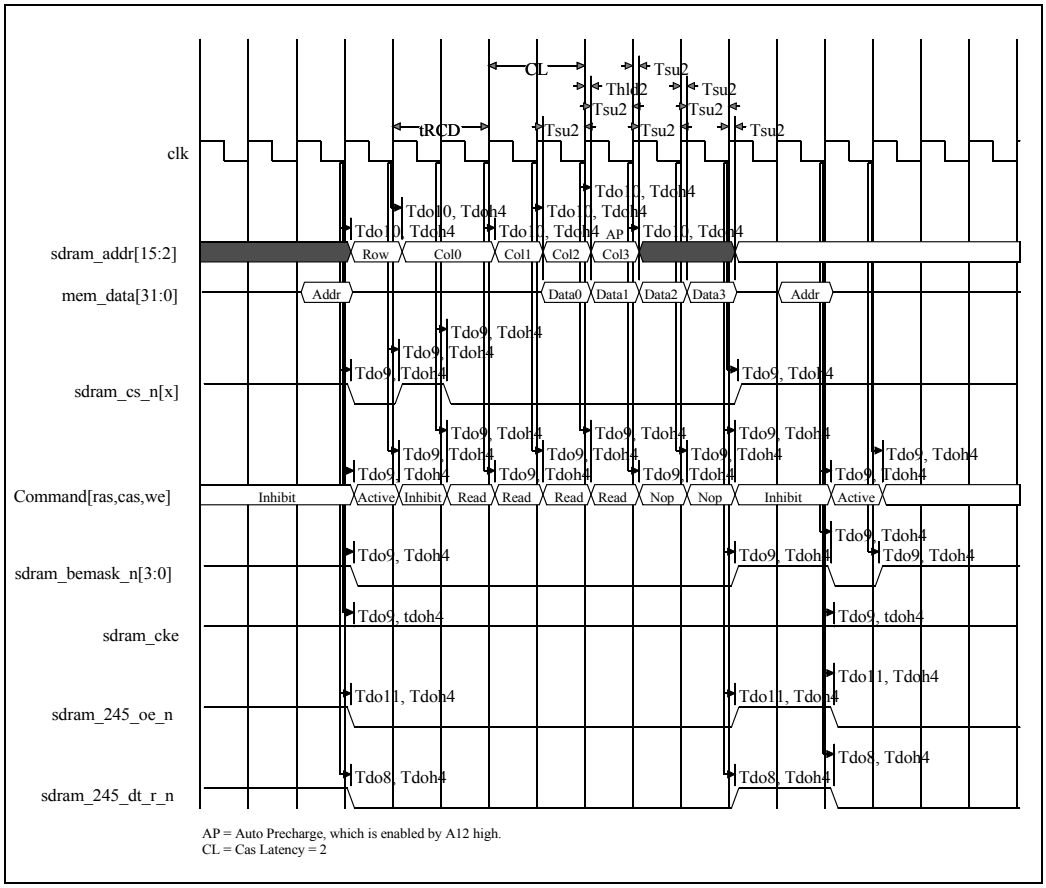


Figure 11.5 SDRAM Non-Page Burst Read

Figure 11.6 shows an SDRAM non-page burst write as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if the page mode is turned off. Because no precharge occurs, the row address is captured immediately and a tRCD active command to r/w command delay—in this case 2 clocks—then occurs. Finally, the column addresses are then captured.

Note that the write data occurs at the same time as its column address and write command. In this case, auto precharge has been programmed, as indicated by the AP symbol, by the col3 sdram_addr. Finally, the beginning of the next transaction is shown. A minimum pre-charge time is enforced, in this case 3 clocks, before the next transaction from the SDRAM controller can begin. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=1, RP=3, RC=8, Status=FF).

Notes

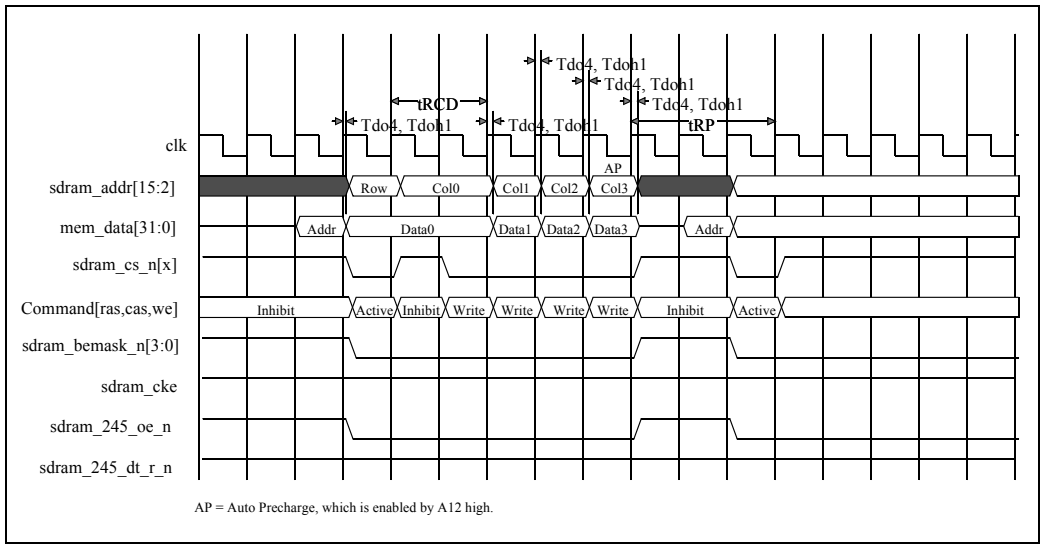


Figure 11.6 SDRAM Non-Page Burst Write

Figure 11.7 shows an SDRAM non-page single word read as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if page mode is turned off. Because no precharge occurs, the row address is captured immediately, a tRCD active command to r/w command delay—in this case 2 clocks—then occurs. Finally, the column address is captured.

Note that there is CAS latency from the column address until the data appears, which is 2 clocks in this case. Also in this case, auto precharge has been programmed—as indicated by the AP symbol—by the col sdram_addr. Finally, the beginning of the next transaction is shown. A minimum pre-charge time occurs coincidentally at least 4 clocks before the next transaction begins, because of the RC32300 CPU core BTA protocol. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=1, RP=4, RC=8, Status=FF).

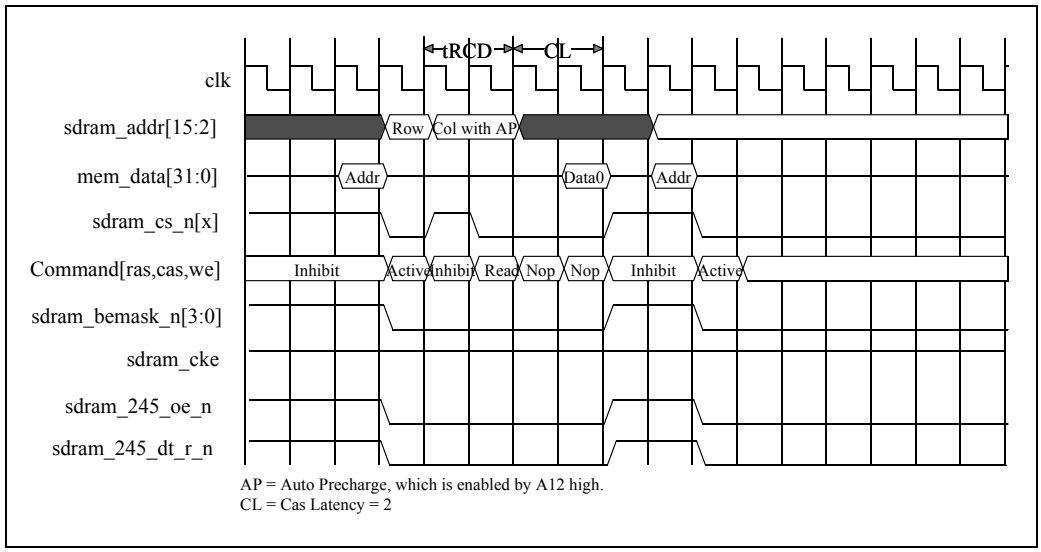


Figure 11.7 SDRAM Non-Page Word Read

Figure 11.8 Shows an SDRAM non-page single word write, as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if the page mode is off. Because no precharge occurs, the row address is captured immediately and a tRCD active command to r/w command delay—in this case 2 clocks— then occurs. Finally, the column address is captured.

Note that the write data occurs at the same time as its column address and write command, as does the sdram_bemask_n[3:0] and dqm bus, indicating which bytes are valid. In this case, auto precharge has been programmed, as indicated by the AP symbol, by the col sdram_addr. Finally, the beginning of the next

Notes

transaction is shown. A minimum pre-charge time is enforced, in this case 3 clocks, before the next transaction from the SDRAM controller can begin. The RP field, in this case 3 clocks, must include both the tRP precharge time and the tWR write recovery time of the SDRAM AC requirements. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=1, RP=3, RC=8, Status=FF).

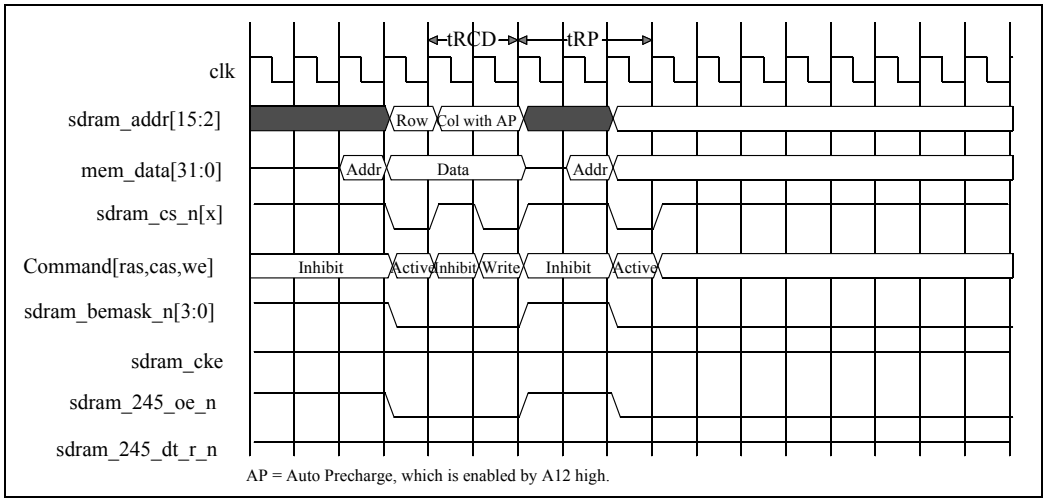


Figure 11.8 SDRAM Non-Page Word Write

Figure 11.9 shows an SDRAM page-hit burst read, as it occurs after the SDRAM controller has been left with an active page open. In this figure, the current memory page matches the previous page. Thus, no precharge occurs. The row address is not needed nor is a tRCD active command to r/w command delay, so the column addresses are captured.

Note that there is CAS latency2 clocks in this case, from the column address until the first data appears. Also in this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col3 sdram_addr. Finally, the beginning of the next transaction is shown. At least 4 clocks coincidently occur before the next transaction because of the RC32300 CPU core BTA protocol, at which time a SDRAM page-hit may reoccur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=2, RC=8, Status=FF).

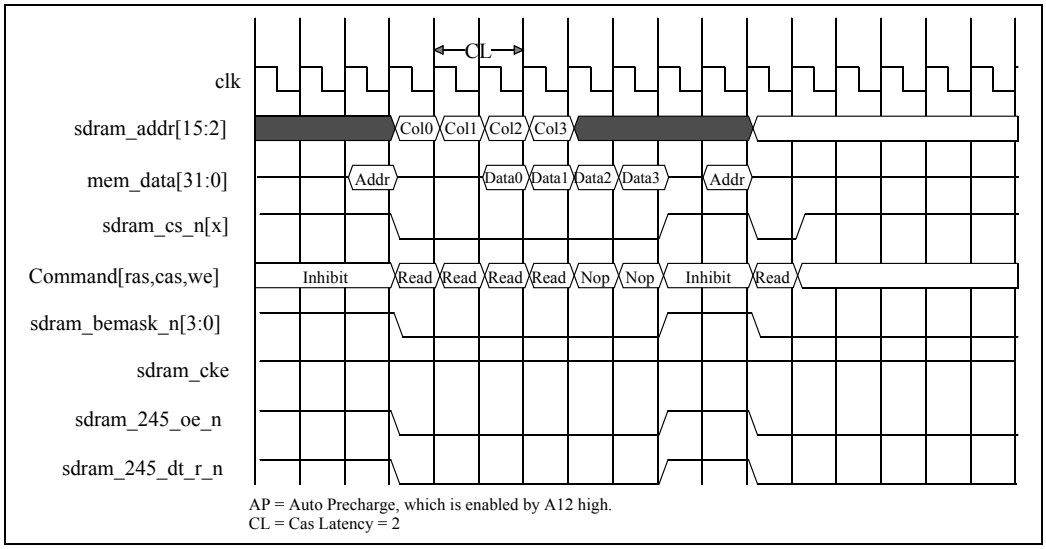


Figure 11.9 SDRAM Page-Hit Burst Read

Figure 11.10 shows a SDRAM page-hit burst write, as it occurs after the SDRAM controller has left with an active page open. In this figure, the current memory page matches the previous page, so no precharge occurs. Neither the row address or tRCD active command to r/w command delay are needed, so the column addresses are captured.

Notes

Note that the write data occurs at the same time as its column address and write command. In this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col3 sdram_addr. Finally, the beginning of the next transaction is shown. At least 2 clocks coincidently occur before the next transaction because of the RC32300 CPU core BTA protocol, at which time a SDRAM page-hit may reoccur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=2, RC=8, Status=FF).

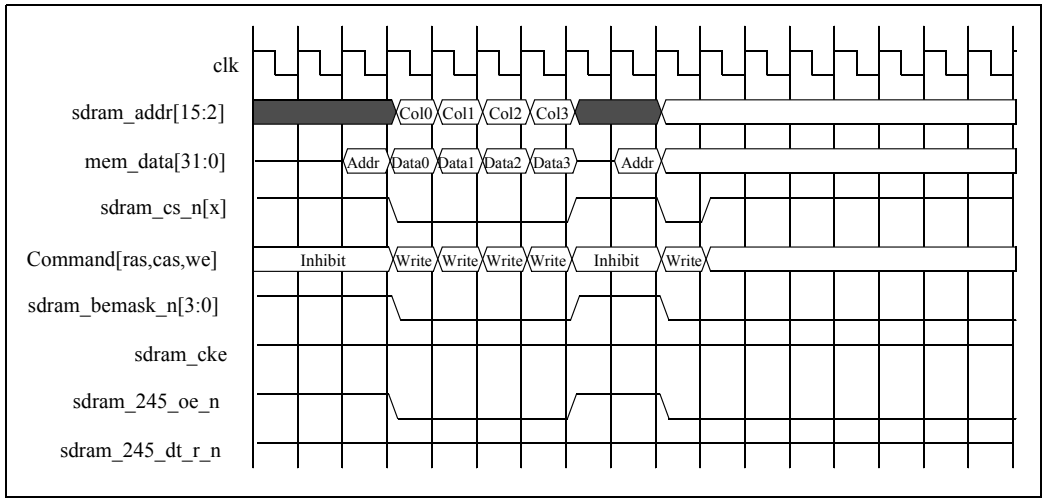


Figure 11.10 SDRAM Page-Hit Burst Write

Figure 11.11 shows a SDRAM page-hit single word read as it occurs after the SDRAM controller has been left with an active page open. In this figure, the current memory page matches the previous page, so no precharge occurs. The row address is not needed nor a tRCD active command to r/w command delay, so the column address is captured.

Note that there is CAS latency, 2 clocks in this case, from the column address until the first data appears. Also in this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col sdram_addr. Finally, the beginning of the next transaction is shown. At least 4 clocks coincidently occur before the next transaction, because of the RC32300 CPU core BTA protocol, at which time an SDRAM page-hit may reoccur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=2, RC=8, Status=FF).

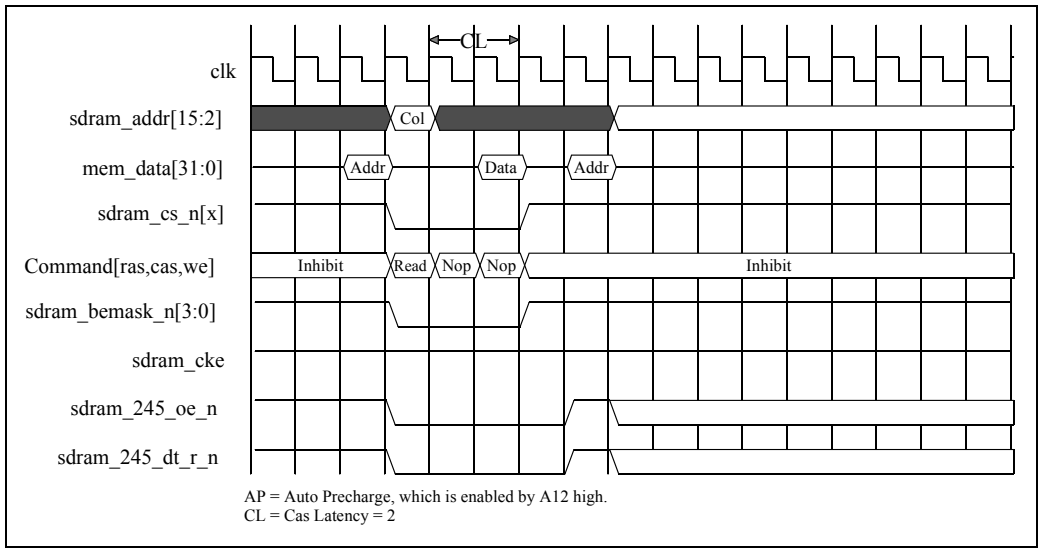


Figure 11.11 SDRAM Page-Hit Word Read

Notes

Figure 11.12 shows an SDRAM page-hit single word write, as it occurs after the SDRAM controller has left with an active page open. In this figure, the current memory page matches the previous page, so no precharge occurs. The row address is not needed nor is a tRCD active command to r/w command delay, so the column address is captured.

Note that the write data occurs at the same time as its column address and write command. In this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col sdram_addr. Finally, the beginning of the next transaction is shown. At least 2 clocks coincidently occur before the next transaction because of the RC32300 CPU core BTA protocol, at which time an SDRAM page-hit may reoccur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=2, RC=8, Status=FF).

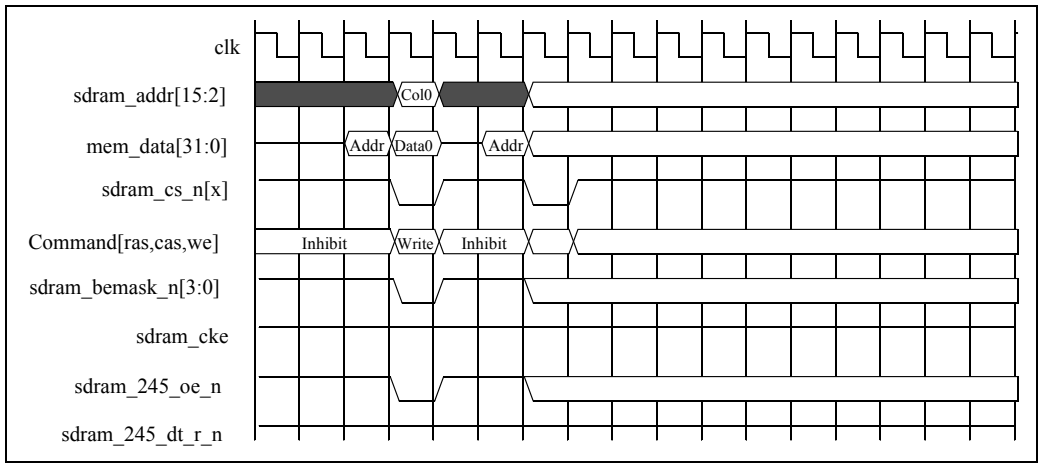


Figure 11.12 SDRAM Page-Hit Word Write

Figure 11.13 shows a SDRAM page-miss burst read, as it occurs after the SDRAM controller has been left with an active page open. In this figure, the current memory page does not match the previous page, so a precharge occurs that is of the Pre-charge programmed length, in this case 3 clocks. Then the row address is captured and a tRCD active command to r/w command delay occurs. The column addresses are then captured.

Note that there is CAS latency, 2 clocks in this case, from the column address until the first data appears. Also in this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col3 sdram_addr. Finally, the beginning of the next transaction is shown. At least 4 clocks coincidently occur before the next transaction because of the RC32300 CPU core BTA protocol, at which time a SDRAM page-miss or hit may (re-)occur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=3, RC=8, Status=FF).

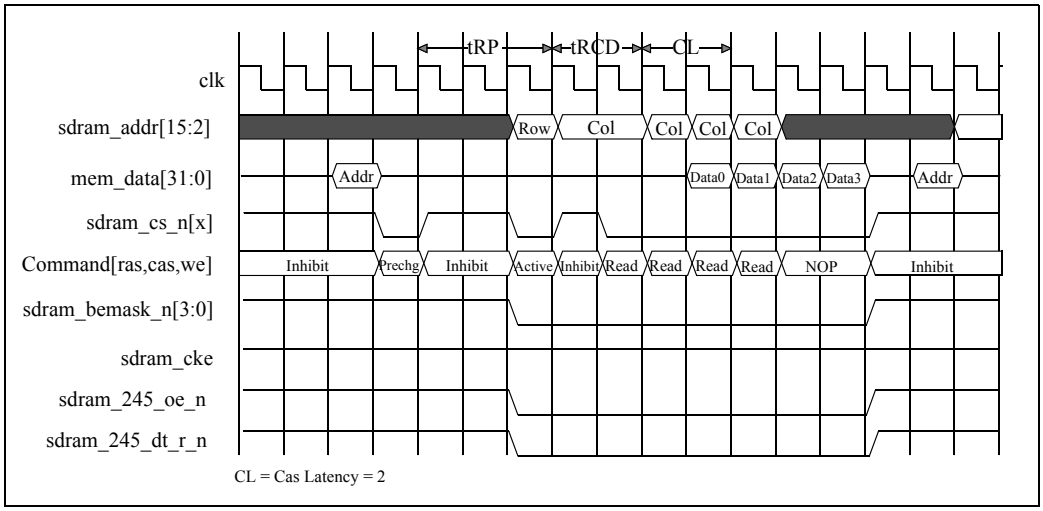


Figure 11.13 SDRAM Page-Miss Burst Read

Notes

Figure 11.14 shows an SDRAM page-miss single word read, as it occurs after the SDRAM controller has been left with an active page open. In this figure, the current memory page does not match the previous page, so a precharge occurs that is of the Pre-charge programmed length, in this case 3 clocks. Then the row address is captured and a tRCD active command to r/w command delay occurs. The column address is then captured.

Note that there is CAS latency, 2 clocks in this case, from the column address until the first data appears. In this case, page left actively asserted has been programmed, as indicated by the lack of an AP symbol, by the col sdram_addr. Finally, the beginning of the next transaction is shown. At least 4 clocks coincidently occur before the next transaction because of the RC32300 CPU core BTA protocol, at which time a SDRAM page-miss or hit may (re-)occur. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=3, RC=8, Status=FF).

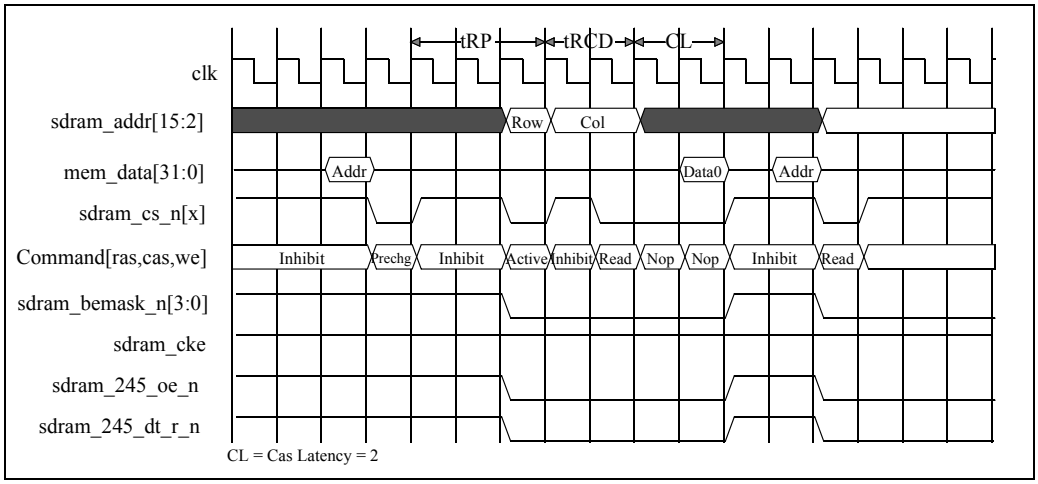


Figure 11.14 SDRAM Page-Miss Word Read

Figure 11.15 shows an 11 clock page mode SDRAM refresh with the Pre-charge Clocks field programmed to the value of 3 clocks and the Refresh Transaction Clocks field programmed to the value of 8 clocks. A minimum of 8 clocks occurs before the next active command can occur, which in this figure is a page mode write.

Note that a non-page mode SDRAM refresh is similar, except that the Pre-charge All command and the Pre-charge Clocks field delay do not occur. Also note that the refresh occurs transparently with respect to concurrent memory controller generated transactions. The refresh will wait until the current SDRAM transaction is complete (if present) and has higher priority over the next/new SDRAM transaction (if present). (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0, RP=3, RC=8, Status=FF).

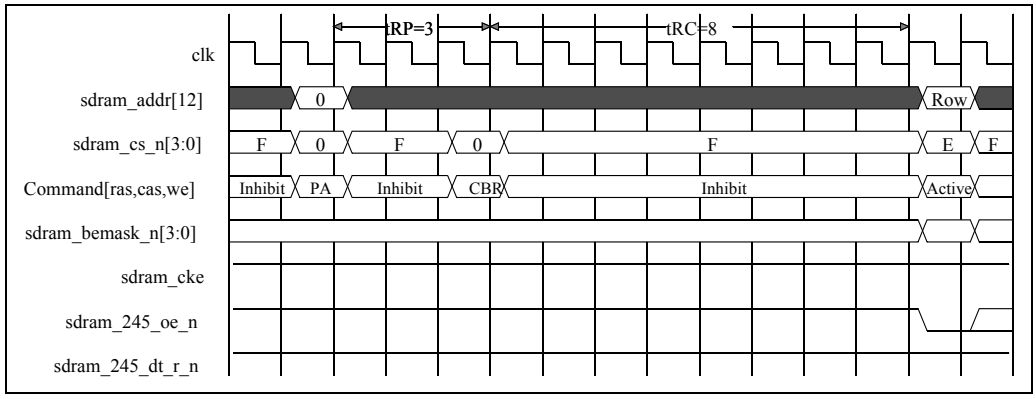


Figure 11.15 SDRAM Refresh

Notes

Connecting the RC32334 to the SDRAMs

Below is the recommended address interface between the RC32334 and the SDRAM banks. A[10] allows "Precharge all banks" during each SDRAM precharge command as well as the appropriate Row address during the row address command.

RC32334	SDRAM Banks
sdram_addr[15:13]	A[13:11]
sdram_addr_12	A[10]
sdram_addr[11:2]	A[9:0]

SODIMM

SODIMM Configuration

RC32334 memory configurations can always use discrete parts. In addition, RC32334 is designed to use memory modules. The RC32334 default memory module configuration is the 100/168-pin DIMM with 4 chip selects. In addition, the RC32334 supports the 144-pin Small Outline DIMM (SODIMM-144) with 2 chip selects.

The use of SODIMM requires two additional pins, sdram_s_n[1:0]. The SODIMM mode requires that the RC32334 SDRAM control register's SODIMM Enable bit be initialized to the SODIMM setting. Note that when the SODIMM mode is enabled, all DIMMs in the system must use the SODIMM signal configuration. In the SODIMM mode, the SDRAM module chip selects are provided on the two sdram_s_n[1:0] signals. sdram_s_n[0] is used to select banks 0 and 1 on the first SODIMM as programmed by the RC32334 address ranges for banks 0 and 1. sdram_s_n[1] is used to select banks 2 and 3 on an optional second SODIMM as programmed by the RC32334 address ranges for banks 2 and 3.

The SODIMM mode also changes the behavior of the sdram_bemask_n[3:0] DQM byte mask enables to only assert on even bank selects, 0 and 2. The SODIMM mode also changes the behavior of the sdram_cs_n[3:0] chip selects to become DQM byte mask enables that only assert on odd bank selects, 1 and 3.

SDRAM SODIMM Even Bank Non-Page Word Read

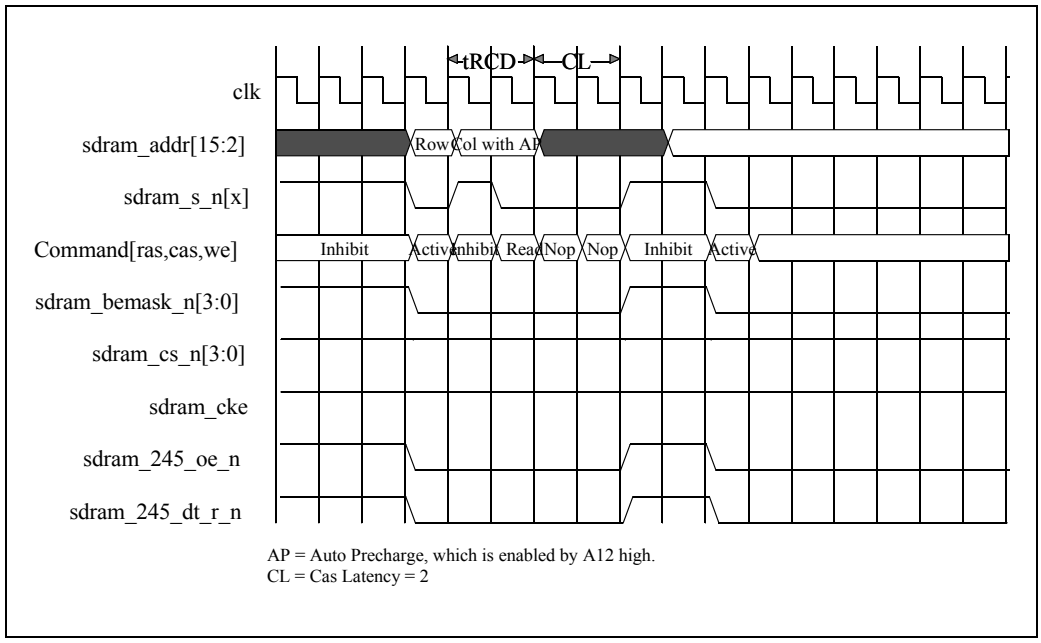


Figure 11.16 SDRAM SODIMM Even Bank Non-page Word Read

Notes

Figure 11.16 shows an SDRAM SODIMM Even Bank non-page word read as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if page mode is turned off. Because no precharge occurs, the row address is captured immediately, a tRCD active command to the r/w command delay (in this example a 2 clock delay) then occurs. Finally, the column address is captured. The module select, sdram_s_n[0] will assert if bank 0 is accessed or sdram_s_n[1] will assert if bank 2 is accessed. The even DQM bus signals, sdram_bemask_n[3:0], are asserted instead of the odd DQM bus signals, sdram_cs_n[3:0]. Note that the burst, write, and page mode accesses for even banks are similar to this case.

SDRAM SODIMM Odd Bank Non-Page Word Read

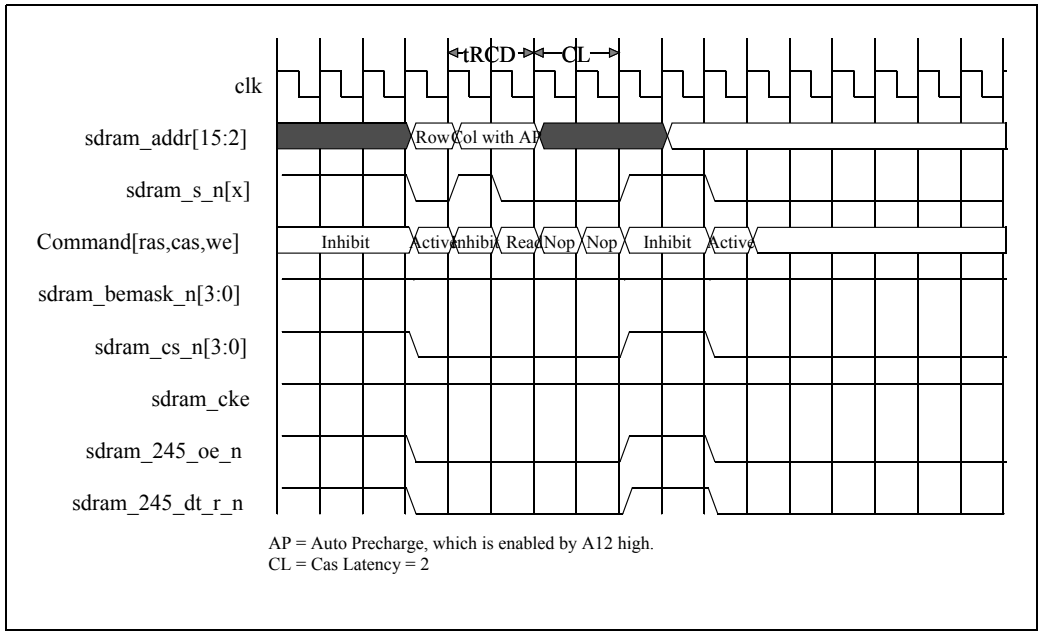


Figure 11.17 SDRAM SODIMM Odd Bank Non-page Word Read

Figure 11.17 shows an SDRAM SODIMM Odd Bank non-page word read as it occurs after the SDRAM controller has been idle, such as after reset, refresh, or if page mode is turned off. Because no precharge occurs, the row address is captured immediately, a tRCD active command to the r/w command delay (in this example a 2 clock delay) then occurs. Finally, the column address is captured. The module select, sdram_s_n[0] will assert if bank 1 is accessed or sdram_s_n[1] will assert if bank 3 is accessed. The odd DQM bus signals, sdram_cs_n[3:0], are asserted instead of the even DQM bus signals, sdram_bemask_n[3:0]. Note that the burst, write, and page mode accesses for odd banks are similar to this case.

Notes

SDRAM SODIMM Refresh

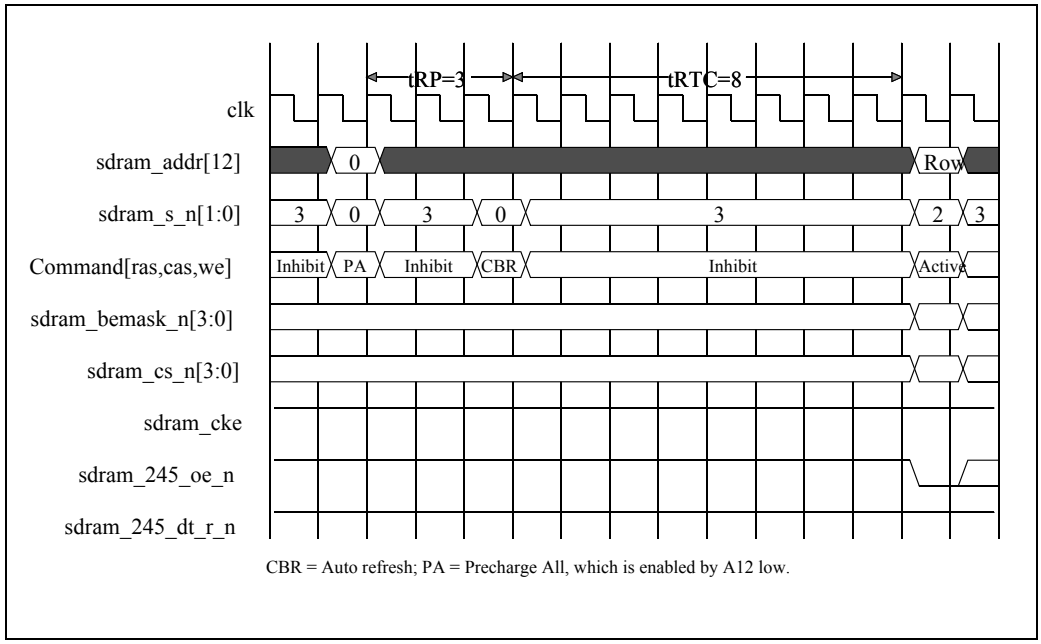


Figure 11.18 SDRAM SODIMM Refresh

Figure 11.18 shows an 11 clock page mode SDRAM SODIMM refresh with the Pre-charge Clocks field programmed to the value of 3 clocks and the Refresh Transaction Clocks field programmed to the value of 8 clocks. A minimum of 8 clocks occur before the next active command can occur, which in this example is a page mode write. Note that a non-page mode SDRAM refresh is similar, except that the Pre-charge All command and the Pre-charge Clocks field delay do not occur.

Also note that the refresh occurs transparently with respect to concurrent memory controller generated transactions. The refresh will wait until the current SDRAM transaction is complete (if present) and has higher priority over the next/new SDRAM transaction (if present). The module selects, sdram_s_n[1:0], are both asserted. (En=1, Mux=01, Size=0, CL=2, RCD=2, AP=0. RP=3, RC=8, Status=FF.) (SODIMM=1.)

output_clk Usage

The RC32334 provides an output_clk output. This clock follows the cpu_masterclk with an approximate 5ns phase delay which aligns the transmit direction control, address, and data signals to a transmit clock. Conventional non-registered PC66, PC100, and PC133 SDRAMs cannot take advantage of the output_clk output feature. Instead, it is recommended that most systems use the cpu_masterclk as the SDRAM clock. Because the output_clk output is enabled at reset time, unless used elsewhere in the system as a transmit aligned clock, output_clk can be turned off to save power using the RC32334 SDRAM control register output_clk Output Enable bit.

Please see the RC32334 Design Considerations document at www.idt.com for the latest SDRAM recommendations, especially concerning the use of transceivers, speed grades, and data width.

Notes

PCI Interface Controller

Notes

Introduction

The PCI Interface Controller complies with *PCI Local Bus Specification, Revision 2.2*[1]. Both master and target modes are supported. The interface implements 3.3V PCI-compliant pads (5V tolerant). The PCI bus operates up to 66 MHz and supports burst transfers. The PCI Interface Controller serves as a PCI bridge between the PCI bus and the RC32334 internal bus. The block diagram of the PCI Interface Controller is shown in Figure 12.1.

The PCI bus interface contains two separate data paths, one for access initiated by the CPU or DMA and one for access initiated by an external PCI agent. Each path has its own FIFO, and each path operates independently from the other.

The PCI controller uses a dedicated DMA engine, separate from the four general purpose DMA controller channels described in Chapter 13, to initiate PCI bus target transfers to and from local memory.

Features

The PCI Interface Module includes the following functions:

- *Master and Target Controllers*
  - *Host or Satellite (Adapter Card) mode*
  - *Capability to access configuration registers from CPU*
  - *Target lock support*
- *PCI Bus Arbitration selection in Host Mode*
  - *Internal arbiter provides:*
    - *Fixed priority or Round Robin*
    - *Arbitrate 3[2] external PCI masters*
  - *Capability to disable internal arbiter to implement arbiter function externally*
- *Mailbox registers*
- *Software programmable endianness byte swapper*
- *Address translation between CPU address space and PCI address space*
- *Independent DMA engine for PCI target transfers from PCI bus to CPU memory*
- *Support for Plug and Play*

PCI Interface Enhancements in Y Silicon Revision

The PCI Interface is one of the modules that has been significantly enhanced in the Y revision of the silicon. TTable 12.1 outlines some of the significant differences between the Z and Y revisions. For more information on the differences between the silicon revisions, refer to Application Note AN-350, RC32334/RC32332 Differences Between Z and Y Revisions, and to the RC32334/RC32332 Device Errata, both posted on IDT's web site at www.idt.com.

1. For operational details and/or timing diagrams not included in this chapter, refer to the PCI 2.2 specification.
2. Arbitrate 2 external PCI masters for the RC32332.

Notes

Function	Z Revision	Y Revision	Comments
PCI specification supported	2.1	2.2	
FIFO size for target read and target writes	8 words deep in each direction.	16 words deep in each direction.	Allows the part to support larger bursts from external PCI bus masters transferring information to/from SDRAM.
FIFO size for master read and master writes	8 words deep in each direction.	Read FIFO is 8 words deep, write FIFO now 16 words deep.	
Maximum burst size for target writes	One word	Eight words	Significant performance improvement for bursts of data from external PCI bus masters.
Number of BAR registers	2	4	Allows users to program certain BARs for SDRAM and others to system control registers, preventing multiple changes to the BAR register. Improves time taken to switch between tasks.
Programmability of BAR address bits	Upper four bits of BAR are programmable.	Upper 24 bits are programmable.	Provides more efficient use of memory space for applications where RC32332/4 is configured for satellite mode operation (example - PCI-add in cards).
Priority of PCI target reads versus writes behavior	Writes favored over reads. Multiple writes can be buffered, but reads cannot. Write FIFO must be flushed before target read is allowed to complete.	Retry of target writes during read operations made optional. Option to perform eight word fetch for one word request.	Opportunity to balance read and write operations.
"Eager prefetch"	Not available	Part will continue to perform target read operations until its FIFO is full.	Significant performance improvement for applications moving large blocks of data across PCI.
TRDY Counter behavior	Holds PCI bus for 16 cycles before retrying a transaction.	Bus can be held longer than 16 cycles.	Violates the PCI specification, but provides opportunity to optimize platform and minimize wasted PCI cycles.

Table 12.1 PCI Differences Between Z and Y Revisions

Notes

To ensure backwards compatibility with Z revision, the new functionality in Y revision is enabled using bits in new registers that were not included in the Z revision. These new registers are shown in Table 12.2 below.

Address	Register
0x1800_20A0	New Feature Register
0x1800_20A4	PCI Target Control Register

Table 12.2 Additional PCI Control Registers

Functional Overview

During reset, three reset initialization pins (mem_addr[22:20]) must be set up properly to select the desired PCI and boot modes for the RC32334. Table 12.3 shows all of the possible mode configurations with the settings of mem_addr[22-20] pins, which are latched after reset. The PCI interface controller can be in either **host mode** or **satellite mode**.

mem_addr[22:21]	mem_addr[20]	Description
0 0	0	Host mode, boot from memory controller, serial EEPROM not supported
0 0	1	Satellite mode, boot from memory controller, serial EEPROM not supported
0 1	0	Not allowed
0 1	1	Satellite mode, boot from PCI serial EEPROM
1 0	x	Reserved
1 1	x	Tri-state memory bus and EEPROM bus during coldreset_n assertion.

Table 12.3 Initialization Pins mem_addr[22:20] Settings

When the PCI Boot Mode option is selected at reset, the PCI host can write to the PCI master enable bit. This is because the PCI Target Not Ready bit (PCI Arbitration Register, bit 2) is cleared when the PCI Boot Mode option is selected.

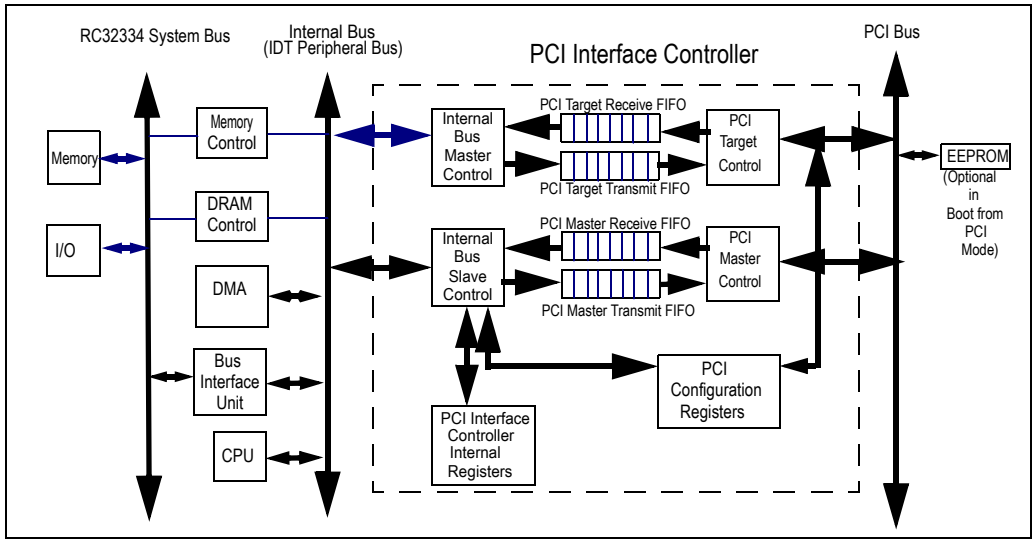


Figure 12.1 PCI Interface Controller Block Diagram

Notes

Memory Mapping

Figures 12.2 and 12.3 show the CPU to PCI memory mapping and the PCI to CPU memory mapping, respectively.

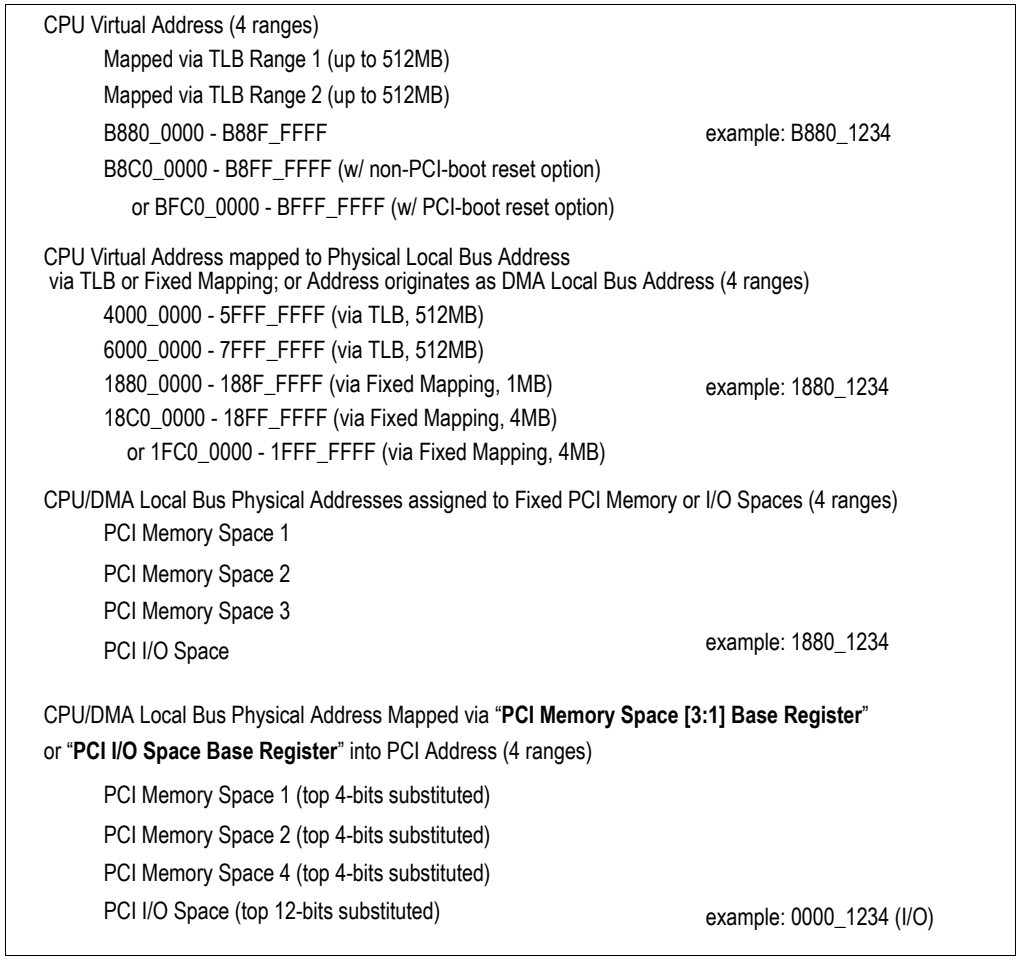


Figure 12.2 CPU to PCI Memory Mapping

Notes

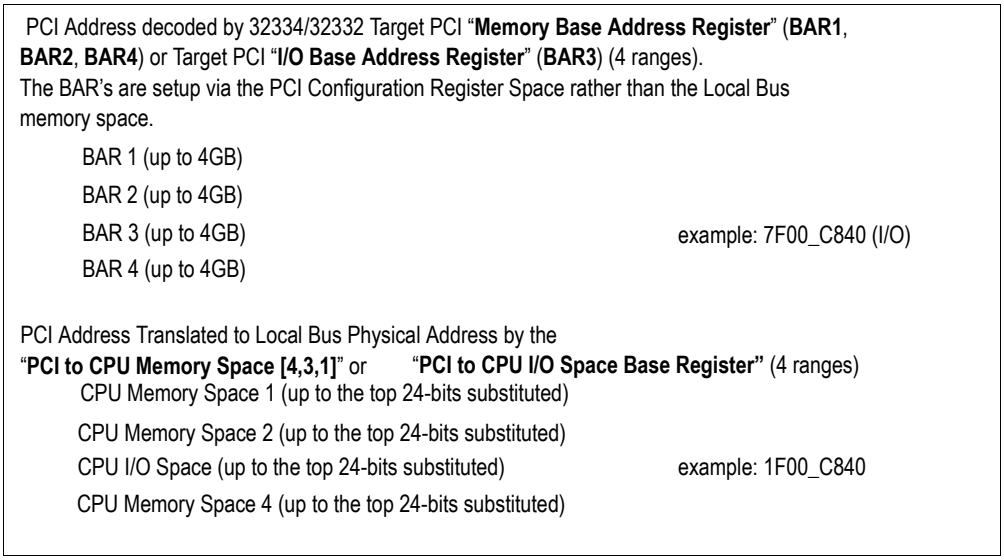


Figure 12.3 PCI to CPU Memory Mapping

RC32334 PCI Bus Target Operation

The PCI interface of the RC32334/RC32332 integrated processors is optimized to support transfers for external PCI bus masters that initiate the transmission/reception of data to/from local SDRAM (termed as target mode in this chapter). When the device is configured for target operation, the module supports up to seven queued target write commands and one queued target read command. When these queues are exceeded, the PCI host transaction will be retried until the command can be accepted. When the RC32334 PCI is configured as a target, external PCI masters can perform up to 8 word bursts.

The PCI target interface allows an external PCI master to be able to read and write any local memory address. This allows an external PCI master to access local SDRAM, Memory-I/O (8, 16 or 32-bit memory) or any internal register. The PCI target interface automatically performs byte scattering (writes) and gathering (reads) for devices on the memory and peripheral bus, and partial writes and reads for SDRAMs. PCI bus accesses to 8-/16-/32-bit external I/O are supported, provided that the I/O addresses are aligned on a word boundary and that the data is located in the correct 1/2/4 byte lanes. Note there is NO byte unpacking. Therefore, for 8- or 16-bit accesses the reminder of the word will not be used.

The PCI bus master can read/write to memory through a CPU memory space 1 BR. The CPU memory space 1 BR translates a PCI address to a local physical CPU address by modifying the top upper 24 bits are programmable. This means that the minimum memory size is 256 Bytes.

Similarly, when accessing I/O peripherals, the CPU I/O space BR translates a PCI address to a local physical CPU address by modifying the top 24 address bits of the PCI address. For I/O accesses, note that only the top 8 address bits of the PCI address bus are used, as the I/O space accessed from the PCI bus is limited to 64 words (256 bytes). The endianness swap setting can be modified for each of the above BRs by setting a bit in each BR.

The RC32334 PCI does not support the cache line wrap mode defined in the PCI specification. Thus, the RC32334 PCI master never generates a cache line wrap mode. A cache line wrap mode cannot be generated from PCI agents, since the RC32334 does not recognize this mode. If this mode is generated from the PCI bus, the RC32334 device will treat the access as linear incrementing.

The PCI bus interface supports target locking. Once a lock has been established, all PCI target transactions to the RC32334/RC32332 device are retried until the lock has been released. Lock operations are useful for creating atomic sequences as seen by external masters on the PCI bus. Lock accesses cannot be issued when the RC32334 is a PCI master.

Note that on PCI target accesses to memory and IO, 0x0xxx_xxxx and 0x0000_00xx spaces are decoded. This enables conventional debug and non-PC system usage of this address space.

Notes

PCI target burst transactions which attempt to burst data beyond the address space allocated to a BAR will terminate with a target disconnect without data. The PCI address spaces mapped by two BAR registers may be contiguous. PCI target burst transactions which attempt to burst data across adjacent address spaces mapped by BAR registers will terminate with a target disconnect without data.

PCI Target Control in the PCI to CPU memory and I/O space base registers contain additional fields beyond the BAR register which control the behavior of the PCI bus interface when acting as a PCI target. These include:

– *A retry timer controls the number of PCI clock cycles the PCI interface will wait (to receive the first data of an access) before it issues a retry command. This is used during target read operations (i.e., memory read, memory read multiple, memory read line, and I/O read) to specify the number of PCI clock cycles the PCI bus interface is allowed to wait (for the first data quantity of a transaction) before the transaction must be retried. During target write operations (i.e., memory write, memory write and invalidate, and I/O write), this field specifies the number of PCI clock cycles the PCI bus interface is allowed to wait (for space to appear in the PCI target input FIFO) before a transaction must be retried. The initial value for the retry timer is specified in the Retry Timer (RTIMER) field of the PCI CPU Memory and I/O Space Base register. Note that PCI 2.2 sets the maximum to 16 PCI clock cycles. However, the RC32334/RC32332 device allows this limit to be extended up to 255 clock cycles. Although this violates the PCI specification, it does provide an opportunity to optimize PCI bandwidth for systems with known PCI-based peripherals.*

– *A disconnect timer controls the number of PCI clock cycles the PCI interface will wait between data transfers. If the PCI bus interface is unable to accept data before the timer expires, the PCI bus will be released. PCI 2.2 specification allows a maximum of 8 PCI clock cycles, but the RC32334/RC32332 allow a value of up to 255 clock cycles.*

The PCI bus interface supports target delayed reads. The PCI bus interface supports only one pending delayed read. If a read is attempted while a delayed read is pending, the transaction is retried and a delayed read is not initiated for the transaction. The external PCI master that initiates a delayed read is expected to retry the transaction until the read completes. The PCI bus interface contains a discard timer. If the master does not repeat a delayed read request within 2^{15} clock cycles, the discard timer will expire and discard the pending read. If the discard timer expires and a pending read is discarded, then the pending read discarded (PRD) bit is set in the PCI controller interrupt pending register. Note that the discard timer can be disabled by setting the disable discard timer (DDT) bit in the PCITC register.

The PCI transaction ordering constraints may be viewed as favoring target write operations, since only a single delayed read is allowed when there are posted writes, while multiple posted writes are allowed when there is a delayed read.

There is also an “eager prefetch” mode. When enabled, the target read PCI block will continue to fetch data until its FIFO is filled. Since the new FIFO is 16 words, this means that a single word fetch can result in 16 words, divided into two bursts, being fetched across the local bus. This will result in substantial throughput improvements in cases of long block reads. However, this mode should be used with caution. If the system is not moving blocks of data, but rather doing isolated reads from specific locations, enabling these features will degrade system performance rather than improve it.

Note that the last 16 words of physical memory should not be accessed by a prefetchable PCI target read. If they are accessed, a system error via the pci_serr_n signal may be signalled since the target read prefetch may try to access non-existent memory beyond the physical memory bank, thereby generating a local bus non-decoded address error.

RC32334 PCI Bus Master Operation

RC32334 PCI Bus Master operation is defined as CPU core or general purpose DMA initiated read/write transfers between the RC32334 and the PCI bus. The address map is shown in Table 12.4 when the CPU core or DMA controller wants to access the PCI bus. The CPU or DMA can read/write to targets on the PCI bus through 3 PCI memory spaces. In PCI master mode, the device can perform quad-word bursts for both read and write accesses. When accessing PCI memory space, the corresponding PCI memory space Base Register (BR) translates a local physical CPU address into a PCI address by modifying the top 4 address bits of the local CPU address.

Notes

Similarly, when accessing PCI I/O space, the PCI I/O space Base Register translates a local physical CPU address into a PCI address by modifying the top 4 address bits of the local CPU address. The BRs can point to the same or overlapping address spaces, if desired. The endianness swap setting can be modified for each of the BRs by setting a bit in each BR.

When the RC32334 PCI is configured as a master, it can perform quad-word burst for both read and write accesses.

From	To	Allocation
1800_2000	1800_2FFF	PCI Internal Registers (4KB)
1880_0000	188F_FFFF	PCI I/O Space (1M)
18C0_0000	18FF_FFFF	PCI Memory Space 3 (4MB) (for non-PCI boot reset option)
1FC0_0000	1FFF_FFFF	PCI Memory Space 3 (4MB) (for boot from PCI bus option)
4000_0000	5FFF_FFFF	PCI Memory Space 1 (512MB)
6000_0000	7FFF_FFFF	PCI Memory Space 2 (512MB)

Table 12.4 PCI Address Map

RC32334 PCI Bus Target Operation

RC32334 PCI Bus Target operation is defined as an external device that initiates a PCI bus read or write transfers between the PCI bus and external memory or between the PCI bus and an I/O peripheral. Note that only 32-bit wide external memory is supported. The PCI bus master can read/write to memory through a CPU memory space 1 BR. The CPU memory space 1 BR translates a PCI address to a local physical CPU address by modifying the top upper 24 bits are programmable. This means that the minimum memory size is 256 bytes.

The PCI controller uses a dedicated DMA engine, separate from the four general purpose DMA controller channels described in Chapter 13, to initiate PCI bus target transfers to and from local memory.

Similarly, when accessing I/O peripherals, the CPU I/O space BR translates a PCI address to a local physical CPU address by modifying the top 24 address bits of the PCI address. Note that only the top 8 address bits of the PCI address bus are used for I/O accesses, as the I/O space accessed from the PCI bus is limited to 64 words (256 bytes). The endianness swap setting can be modified for each of the above BRs by setting a bit in each BR.

The RC32334 PCI Bus Target supports PCI bus accesses to 8-/16-/32-bit external I/O, assuming the I/O addresses are aligned on a word boundary and that the data is located in the correct 1/2/4 byte lanes. Note there is NO byte unpacking. Therefore 8- or 16-bit access the reminder of the word will not be used.

When the RC32334 PCI is configured as a target, external PCI masters can only perform up to 8 word bursts. If the write address is such that it will cross a 1024-byte boundary (minimum SDRAM page size), the current write will end when the 0x3FC offset is reached and a new IPBus write is ready to begin.

The RC32334 PCI does not support the cache line wrap mode defined in the PCI specification. Thus, the RC32334 PCI master never generates a cache line wrap mode. A cache line wrap mode cannot be generated from PCI agents, since the RC32334 does not recognize this mode. If this mode is generated from the PCI bus, the RC32334 device will treat the access as linear incrementing.

The RC32334 PCI controller supports lock accesses when RC32334 is a PCI target. However, the lock accesses cannot be issued when the RC32334 is a PCI master.

Note that on PCI target accesses to memory and IO, 0x0xxx_xxxx and 0x0000_00xx spaces are decoded. This enables conventional debug and non-PC system usage of this address space.

PCI Satellite Mode

The PCI bus interface can also be configured for satellite mode operation. The satellite mode can be initiated in two ways:

Notes

- *The satellite can boot from the Memory Controller. In this case, the bootstrapping code for the satellite resides in the local memory space from which the satellite board boots up*
- *The satellite can boot from the PCI serial EEPROM. In this case, the satellite loads its configuration registers from a serial EEPROM and then attempts to boot over the PCI bus.*

In either case, the host PCI bridge in the system is required to program the PCI configuration registers prior to the satellite generating or receiving any PCI cycles on the PCI bus.

To ensure the correct Satellite mode of operation, the System Controller needs to configure mem_addr[22:20] bits on reset. When mem_addr[22:20] is configured to [001], the satellite is set to boot from the Memory Controller. When mem_addr[22:20] is configured to [011], the satellite is set to boot from the PCI serial EEPROM.

Booting from the Memory Controller

Booting from the Memory Controller, the Satellite mode receives and generates PCI cycles on the PCI bus. The initialization steps are as follows:

1. Configure the local boot ROM on the satellite system to:
   - *Link local PCI registers and CPU (PCI to CPU and CPU to PCI)*
   - *Set up the PCI configuration register Master Latency Timer, Cacheline Size, Retry Timeout, TRDY Timeout, etc.*
   - *Reset the PCI Target Not Ready bit in the PCI Arbitration Register.*
2. Configure the satellite PCI Configuration Registers using the host PCI bridge:
   - *Memory Base Address Register (Configuration Header Offset: 0x10)*
   - *I/O Base Address Register (Configuration Header Offset: 0x18)*
   - *Enable the Bus Master, Memory, and I/O Access in the PCI Configuration Command Register.*

Booting from the PCI Serial EEPROM

The Satellite mode, booting from the PCI serial EEPROM, loads the PCI configuration registers from the serial EEPROM. The initialization steps are as follows:

1. Program the serial EEPROM with the desired configuration register values.
2. Configure the satellite PCI configuration registers using the host PCI bridge.
   - *Memory Base Address Register (Configuration Header Offset: 0x10)*
   - *I/O Base Address Register (Configuration Header Offset: 0x18)*
   - *Enable the Bus Master, Memory, and I/O Access in the PCI Configuration Command Register.*

Once the satellite PCI interface is enabled by the host PCI bridge by writing to the Command Configuration Register, the satellite generates an Instruction Fetch cycle with the local bus physical address 0x1FC0 0000. This address is translated to the PCI bus address 0x0FC0 0000 before being placed on the PCI bus by the satellite's local PCI Memory Space 3 Base Register, its contents being all 0's on reset.

The satellite can only boot from a 32-bit port-width external device sitting across the PCI bus. The target device selected for the PCI address 0x0FC0 0000 must have a 32-bit boot memory in this address space (typically a 32-bit EPROM space or an SDRAM space where the bootstrap code for the satellite is placed prior to enabling the satellite). The Target Not Ready bit in the PCI Arbitration Register is reset by default. Also, the BusError is disabled at power up in this mode. The BusError must be enabled by the startup code as soon as the satellite is initialized in order to catch any non-decodable address cycles on the PCI bus.

In the PCI-boot mode, the System Controller Internal BIU BusError Register has the CPU BusError, IP BusError, and Watch Dog timeout bits disabled, which allows the RC32334 to wait indefinitely for the PCI host to initialize the system.

Serial EEPROM Interface

When booting from PCI, the serial EEPROM is used to load the PCI configuration header in the Satellite mode.

The boot serial EEPROM must be compatible with and at least as large as the NM93CS46 (1024-bit or greater), which uses the MICROWIRE™ of National Semiconductor serial protocol. The RC32334 will sequentially read each of the register addresses listed in Table 12.5, starting from EEPROM address 0x00, skipping unused addresses, and continuing until EEPROM address 0x3E. Each EEPROM address corre-

Notes

sponds to a 16-bit datum (not the 8-bit datum that PCI address uses), such that each EEPROM address holds a 16-bit PCI field. Thus, all odd EEPROM addresses are unused by the RC32334 PCI EEPROM interface and can be used for other storage purposes. The 16-bit PCI fields correspond to the definitions of the corresponding PCI Configuration Registers.

Field Name	EEPROM Address
Device ID	0x00
Vendor ID	0x02
Status	0x04
Class Code (MSB's)	0x08
Class Code (LSB), Revision ID	0x0A
Header Type	0x0C
Subsystem ID	0x2C
Subsystem Vendor ID	0x2E
Min_Lat, Min Gnt	0x3C
Interrupt Pin	0x3E

Table 12.5 PCI Serial EEPROM Address Fields

PCI Commands Supported

The RC32334 PCI master supports PCI memory read line and memory write invalidate commands. Memory read line performs a quad-word burst read and memory write invalidate performs a quad-word write. To enable the memory write invalidate command, the cache line size in the Cacheline Size Configuration Register must be nonzero (see Figure 12.21), the memory write and invalidate enable bit in the Command Configuration Register (see Figure 12.17) must be enabled, and a burst write must be generated from the CPU or, more typically, from the DMA. As a PCI target, the RC32334 supports memory read line, memory read multiple, and memory write invalidate.

Table 12.6 summarizes the PCI command codes supported (and not supported) by the controller as master and as target.

CBEn[3:0]	Command	As a Master	As a Target
0000	Interrupt Acknowledge	No	Ignored
0001	Special cycle	No	Ignored
0010	I/O read	Yes	Yes
0011	I/O write	Yes	Yes
010x	Reserved		
0110	Memory read	Yes	Yes, prefetch 4 words
0111	Memory write	Yes	Yes
100x	Reserved		
1010	Configuration read	Yes	Yes
1011	Configuration write	Yes	Yes
1100	Memory read multiple	No	Yes, prefetch 8 words

Table 12.6 PCI Commands (Part 1 of 2)

Notes

CBEn[3:0]	Command	As a Master	As a Target
1101	Dual address cycle	No	Ignored
1110	Memory read line	Yes, quad word burst read	Yes, aliased to memory read
1111	Memory write and invalidate	Yes, quad word burst write	Yes, aliased to memory write

Table 12.6 PCI Commands (Part 2 of 2)

PCI Configuration Register Access

The way RC32334 interfaces and accesses the configuration registers is defined in the PCI specification 2.1, Section 3.7.4.1, Configuration mechanism #1. This mechanism requires the following two RC32334 internal registers be defined to access PCI configuration space:

- *PCI Configuration Address Register at 1800_2cf8*
- *PCI Configuration Data Register at 1800_2cfc.*

A PCI configuration register should be accessed in the following manner:

a. Write the desired address of a configuration register to the PCI Configuration Address register

b. Read from (or write to) the PCI Configuration Data register to receive (or to send) data.

The data in the PCI Configuration Data register will be automatically received from (or sent to) the desired configuration register. The device number field of the PCI Configuration Address Register is used to select the IDSEL line of the PCI satellite to be configured. See Table 12.7 below.

Device Number	Address Line	Device Number	Address Line	Device Number	Address Line
0x00	Internal access	0x08	pci_ad[18]	0x10	pci_ad[26]
0x01	pci_ad[11]	0x09	pci_ad[19]	0x11	pci_ad[27]
0x02	pci_ad[12]	0x0A	pci_ad[20]	0x12	pci_ad[28]
0x03	pci_ad[13]	0x0B	pci_ad[21]	0x13	pci_ad[29]
0x04	pci_ad[14]	0x0C	pci_ad[22]	0x14	pci_ad[30]
0x05	pci_ad[15]	0x0D	pci_ad[23]	0x15	pci_ad[31]
0x06	pci_ad[16]	0x0E	pci_ad[24]	—	—
0x07	pci_ad[17]	0x0F	pci_ad[25]	—	—

Table 12.7 PCI Device to IDSEL Mapping

Device Number 0x00 refers to the PCI host (RC32334) in which case the transaction is handled internally and the PCI Bus remains idle. Device Numbers 0x15 to 0x01 will assert a single pci_ad[31:11] line high during the configuration access shown in Table 12.7. The PCI system board is assumed to resistively couple the appropriate pci_ad[31:11] line to each satellite's pci_idsel line.

Before the RC32334 can be ready to perform any PCI operations, its PCI configuration registers must be set up correctly. The RC32334 PCI master and target are defaulted to not ready (disabled) after reset.

If the RC32334 PCI is in host mode, then the CPU needs to configure the RC32334 PCI configuration registers, including read-only configuration registers. The RC32334 PCI target is not ready until the PCI target not ready bit (bit 2 of the PCI Arbitration Register) is set to 0. When the RC32334 PCI target is not ready, all the PCI assesses to RC32334 from the PCI bus will be retried by the PCI controller. Thus, after the writing of configuration registers is complete and RC32334 is ready, bit 2 of the arbitration register needs to be set to 0 to enable the RC32334 PCI target operations.

Notes

When writing the configuration registers, the RC32334 in host mode will perform 5 extra cycles of address stepping, such that the PCI address is valid for 5 clocks before PCI_frame_n is asserted. This allows the target to resistively couple an address signal to its pci_idsel pin.

If the RC32334 PCI is in satellite mode, read-only configuration registers can be loaded by the CPU core. If the CPU core finishes loading the read-only configuration registers in the satellite mode, then bit 2 of the PCI Arbitration Register needs to be set to 0, so that the RC32334 PCI target can respond to accesses from the PCI bus. If the boot mode initialization chooses to use the EEPROM to load read-only configuration registers, then the system using the RC32334 will be booted from the PCI bus after reset, instead of from the normal local bus address space.

To enable RC32334 PCI master operation, the enable bus master bit in the configuration command register must be set to 1 either by the CPU core if the RC32334 PCI is in host mode or by an external PCI host if in satellite mode.

PCI Polling Error Handling

When the RC32334 device issues a config_read cycle to an unpopulated PCI slot, the device should read back 0xFFFFFFFF. The RC32334 can also be configured to ignore PCI bus errors. This is controlled through bit 7 in the Bus Interface Unit (BIU) BUSErr Control Register. Even when buserror is disabled, a bus error interrupt is still generated which can be polled by PCI BIOS software.

PCI Interrupts

If the PCI bus writes a 1 to one of the low order 4 bits in the PCI_to_CPU mailbox pending register, then a corresponding interrupt is generated to the CPU core via the internal cpu_int_n[3] signal and the CPU core must service and clear this interrupt. (For testing purposes only, the CPU may also set the interrupt.) If the CPU writes a 1 to one of the low order 4 bits in the CPU_to_PCI pending mailbox register, then a corresponding interrupt is generated to the PCI bus via the pci_inta__n pin, and this interrupt needs to be cleared from the PCI bus. Note that the PCI_to_CPU mailbox interrupt can be generated in either host or satellite mode, while CPU_to_PCI mailbox interrupts can be generated only in the satellite mode.

The CPU core or DMA can initiate a PCI access and know whether it is failed or not by enabling both the PCI master read error interrupt and the PCI master write error interrupt defined in the PCI controller interrupt pending register. Note that both interrupts must be enabled to ensure that a RC32334 PCI master access error can be observed. If only one of the interrupts is enabled, then a master access error may not be detected.

To enable any PCI address or data parity error detection by the PCI interface controller, both the parity error response bit and SERR# enable bit must be enabled in the command configuration register. Two kinds of parity errors can be reported to the CPU by using two specific interrupts. These two errors are PCI Target Write Data Parity Error, and PI Master Data Parity Error, as indicated in the PCI controller interrupt pending register.

Signal Definitions

Note that the pci_serr_n I/O signal to the RC32334 is connected as an output, but the signal is not connected internally inside the device as an input. Users wishing to utilize this signal should connect this signal externally to either the cpu_nmi_n signal or a high priority interrupt line on the PCI host. Additionally, a pci_eeprom_cs signal has been added as a PIO pin. This enables external EEPROMs, configured in the PCI memory address space, to be written to and reprogrammed. To support the feature, an extra PIO register has also been added. Note that the I/O direction of pci_gnt_n[1] is controlled by the PIO Direction Register, not by the PCI arbiter mode. See Chapter 15, Programmable I/O (PIO) Controller.

When the RC32334 is in PCI satellite mode, the pci_gnt_n[2:0] and pci_req_n[2:0] pins on the RC32334 each have a different name and use[1]. Table 12.8 shows the name and the direction of each pin for the different settings of RC32334. A complete description of all PCI signals is provided in Chapter 1, RC32334 Device Overview.

[1] Depending on the PCI Mode for which the default is configured.

Notes

	RC32334 in host mode			RC32334 in satellite mode
	Reset	Use internal arbiter	Use external arbiter	
pci_gnt_n[2], output	tri-stated if host	pci_gnt_n[2], output	not used, tri-stated	pci_inta_n, open-collector output
pci_gnt_n[1], bidirectional[1]	output	pci_gnt_n[1], output	not used, output	pci_eeprom_cs, output
pci_gnt_n[0], bidirectional	tri-stated	pci_gnt_n[0], output	pci_req_n=output	pci_gnt_n, input
pci_req_n[2], input	tri-stated	pci_req_n[2], input	not used, tri-stated	pci_idsel, input
pci_req_n[1], input[2]	tri-stated	pci_req_n[1], input[2]	not used, tri-stated	not used, tri-stated
pci_req_n[0], bidirectional	tri-stated	pci_req_n[0], input	pci_gnt_n, input	pci_req_n, output

Table 12.8 RC32334 Muxed PCI Pin Names and Directions

1. pci_gnt_n[1] output enable control is determined by the PIO Pin Direction Register bit field, which defaults to the Output Direction at reset.

2. There is no pci_req_n[1] in the RC32332.

Register Definitions

Base Address	Register Function	Offset Address	Effective Address
1800_0500	PCI Controller Interrupt Pending Register 11	B0	Base + Offset
1800_0500	PCI Controller Interrupt Mask Register 11	B4	
1800_0500	PCI Controller Interrupt Clear Register 11	B8	
1800_0500	CPU to PCI Mailbox Interrupt Pending Register 12	C0	
1800_0500	CPU to PCI Mailbox Interrupt Mask Register 12	C4	
1800_0500	CPU to PCI Mailbox Interrupt Clear Register 12	C8	
1800_0500	PCI to CPU Mailbox Interrupt Pending Register 13	D0	
1800_0500	PCI to CPU Mailbox Interrupt Mask Register 13	D4	
1800_0500	PCI to CPU Mailbox Interrupt Clear Register 13	D8	
1800_2000	PCI New Feature Register	0A0	
1800_2000	PCI Target Control Register	0A4	
1800_2000	PCI Memory and I/O Space 1 Base Register	0B0	
1800_2000	PCI Memory and I/O Space 2 Base Register	0B8	
1800_2000	PCI Memory and I/O Space 3 Base Register	0C0	
1800_2000	PCI Memory and I/O Space 4 Base Register	0C8	
1800_2000	PCI Arbitration Register	0E0	
1800_2000	PCI CPU Space1 Base Register	0E8	
1800_2000	PCI CPU Space 2 Base Register	0F4	
1800_2000	PCI CPU Space 3 Base Register	100	

Table 12.9 PCI Interface Control Register Address Map (Part 1 of 2)

Notes

Base Address	Register Function	Offset Address	Effective Address
1800_2000	PCI CPU Space 4 Base Register	10C	
1800_2000	PCI Configuration Address Register	CF8	
1800_2000	PCI Configuration Data Register	CFC	

Table 12.9 PCI Interface Control Register Address Map (Part 2 of 2)

Note: A detailed description of interrupt related registers is provided in Chapter 14, Expansion Interrupt Controller.

PCI Controller Interrupt Pending Register 11

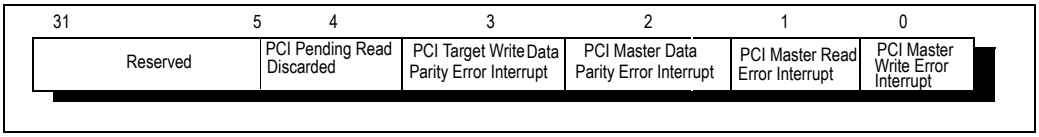


Figure 12.4 PCI Controller Interrupt Pending Register 11 Fields

Bit	Field	Description
31:5	Reserved	
4	PCI Pending Read Discarded (PRD)	Allow Interrupt from a PCI Target read to be discarded as the discard timer expired.
3	PCI Target Write Data Parity Error Interrupt	Interrupt due to a data parity error of a target write to the RC32334 PCI
2	PCI Master Data Parity Error Interrupt	Interrupt due to a data parity error of a RC32334 PCI master read or write
1	PCI Master Read Error Interrupt	Interrupt indicating a failed PCI master access, which may be possibly caused by a PCI master read
0	PCI Master Write Error Interrupt	Interrupt indicating a failed PCI master access, which may be possibly caused by a PCI master write

Table 12.10 PCI Controller Interrupt Pending Register 11 Field Descriptions

CPU to PCI Mailbox Interrupt Pending Register 12

Setting a bit in the CPU to PCI mailbox interrupt pending register by the CPU will generate a corresponding interrupt to the PCI bus.

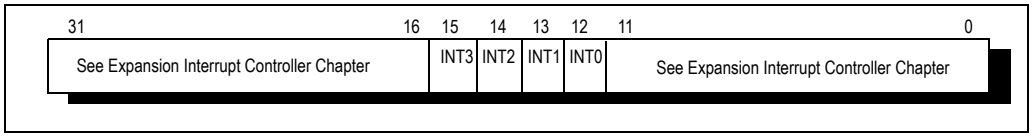


Figure 12.5 CPU to PCI Mailbox Interrupt Pending Register 12 Fields

Notes

Bit	Field Name	Description
31:16	See Chapter 14, Expansion Interrupt Controller.	
15	Interrupt 3	0 = No interrupt (default) 1 = Interrupt pending
14	Interrupt 2	0 = No interrupt (default) 1 = Interrupt pending
13	Interrupt 1	0 = No interrupt (default) 1 = Interrupt pending
12	Interrupt 0	0 = No interrupt (default) 1 = Interrupt pending
11:0	See Chapter 14, Expansion Interrupt Controller.	

Table 12.11 CPU to PCI Mailbox Interrupt Pending Register 12 Field Descriptions

PCI to CPU Mailbox Interrupt Pending Register 13

External PCI Bus Masters may access the PCI to CPU mailbox interrupt pending register via a RC32334 Target memory or I/O access. This assumes that either the PCI CPU memory space base register or the PCI CPU I/O space base register is set up to allow access to the RC32334 System Controller physical address range base of 0x18000000.

Setting a bit in the PCI to CPU mailbox interrupt pending register by the PCI bus will generate a corresponding interrupt to the CPU bus.

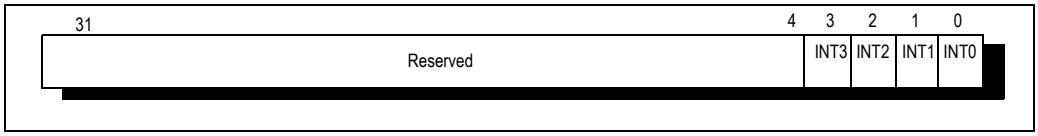


Figure 12.6 PCI to CPU Mailbox Interrupt Pending Register 13 Fields

Bit	Field Name	Description
31:4	Reserved	
3	Interrupt 3	0 = No interrupt (default) 1 = Interrupt pending
2	Interrupt 2	0 = No interrupt (default) 1 = Interrupt pending
1	Interrupt 1	0 = No interrupt (default) 1 = Interrupt pending
0	Interrupt 0	0 = No interrupt (default) 1 = Interrupt pending

Table 12.12 PCI to CPU Mailbox Interrupt Pending Register 13 Field Descriptions

PCI Memory Space [1,2,3] Base Register

Whenever PCI Memory is accessed from the CPU or DMA, the high order 4 bits of the CPU physical address are replaced by bits 31:28 of this register to generate the PCI address.

Notes

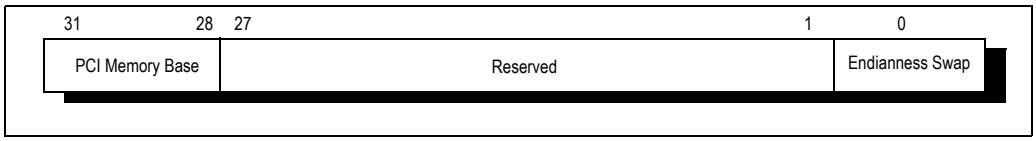


Figure 12.7 PCI Memory Space [1,2,3] Base Register

Bit	Field Name	Description
31:28	PCI Memory I/O Base	Default to 0 These 4 bits replace the top 4 bits of the CPU physical address
27:1	Reserved	0
0	Endianness Swap	Value Description 1 = Byte swap 0 = No byte swap (default)

Table 12.13 PCI Memory Space [1,2,3] Base Register Field Descriptions

PCI I/O Base Register

Whenever I/O space is accessed from the CPU or DMA, the high order 12 bits of the CPU physical address are replaced by bits 31:20 of this register to generate the PCI address.

Note that if compatibility with existing software written for the RC32134 system controller is desired, this register should be programmed with 0x _ 88 _ _ _ _ h.

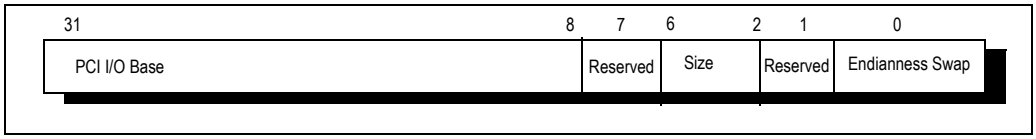


Figure 12.8 PCI I/O Base Register

Bit	Field Name	Description
31:8	CPU Memory or I/O Base	Default value is 0. Up to the top 24 bits translate/replace the top 24 bits of the PCI address with a Local Bus address. The Size Field determines the number of bits to translate/replace.
7	Reserved	Reserved to '0'.

Table 12.14 PCI I/O Base Register Field Descriptions (Part 1 of 2)

Notes

Bit	Field Name	Description
6:2	Size	Address Space Size. This field indicates the size of the address space for the corresponding PCI base address register and the number of the CPU Memory or I/O Base bits to translate/replace. All bits greater than or equal to Size in the Memory or I/O Base Address Register (BAR) may be modified. Bits less than Size and greater than or equal to bit 4 always return a value of zero when read and cannot be modified. Setting the Size field to a value less than 8 disables the PCI base address register from decoding and causes the appropriate BAR bits to always return a value of zero when read and cannot be modified. One may also view this field as indicating the number of the Most Significant bit to be used in the decode, the minimum number being the 8th upper most significant bit. All bits greater than or equal to Size in the CPU Memory or I/O Base field are used to translate/replace the top bits of the PCI address with a Local Bus address. **Value** / **Description** >= 8 — Set size to 2^{SIZE} 28 — Set size to 2^{28} 7 to 1 — Set size to 0, i.e., disabled 1 — Set size to 0, i.e., disabled (default for BAR2 and BAR4) 0 — Set size to 2^{28} (default for BAR1 and BAR3)
1	Reserved	Reserved to '0'.
0	Endianess Swap	This bit controls byte swapping for PCI transactions that map to the local bus through the BAR register. **Value** / **Description** 1 — Byte swap 0 — No byte swap (default)

Table 12.14 PCI I/O Base Register Field Descriptions (Part 2 of 2)

New Feature Register

This register is not present in the Z revision of the RC32334/RC32332 devices. To ensure backwards compatibility, new functionality was provided by adding new registers. Upon system boot-up, this register defaults to provide compatibility with Z revision silicon as follows:

- *bit 1 is zero, enabling configuration read cycles to generate an interrupt should a PCI error occur*
- *bit 0 is zero, enabling bits 23:20 of the PCI Base Register to program higher order bits.*

When a config_read cycle is generated, a PCI error will produce read data as 0xFFFFFFFF. With the PCI Config Read Suppress Bus Error bit field set in the PCI New Feature Register, the PCI Interface Controller suppresses the generation of an IPBus Error for Config Read errors and returns the read data as 0xFFFFFFFF. Thus, neither a bus error exception to the CPU nor an IPBus error interrupt will occur. Even with this bit set, Non-Config Reads to conventional PCI memory space still signal a CPU bus error.

Notes

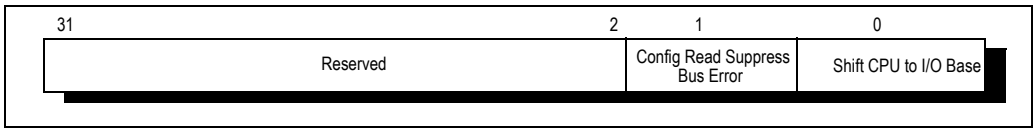


Figure 12.9 PCI New Feature Register

.

Bits	Field Name	Description
31:2	Reserved	
1	Config Read Suppress Bus Error	.
0	Shift CPU to I/O Base Field	.

Value	Description
1	Suppress internal IPBus error generation on PCI Config Read errors.
0	IPBus Bus Error generated on PCI Config Read errors (default).

Value	Description
1	Use the PCI to CPU I/O Base Register Base Address field and Size field to program bits 31:8 (which corresponds to bits 31:8).
0	Use bits 23:20 of the PCI to CPU I/O Base Register field to program bits 31:28 (requires the PCI to CPU I/O Base Register Size Field to be set to 2^{28}).

Table 12.15 PCI New Feature Register Field Descriptions

PCI Target Control Register

A new register for the Y revision of silicon, PCI Target Control Register, is added at physical address 0x1800_20A4.

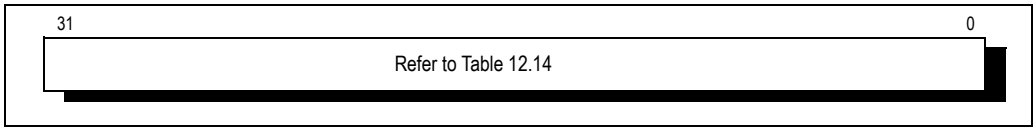


Figure 12.10 PCI Target Control Register

Notes

Bits	Field Name	Description
31	Reserved	Reserved to '0'.
30	Eager Prefetch for BAR4	Eager Prefetch mode. On a Memory Read line or a Memory Read Multiple command decoded by Memory Base Address Register 4 (Bar4), after the initial prefetch, if the Target Read continues without a disconnect and the Target Read FIFO has at least 8 data words empty, then prefetch the next block of data. **Value** / **Description** 1 – Use Eager Prefetch mode. 0 – Do not use Eager Prefetch mode (default).
29	Reserved	Reserved to '0'.
28	Eager Prefetch for BAR2[1]	Eager Prefetch mode. On a Memory Read line or a Memory Read Multiple command decoded by Memory Base Address Register 2 (BAR2), after the initial prefetch, if the Target Read continues without a disconnect and the Target Read FIFO has at least 8 data words empty, then prefetch the next block of data. **Value** / **Description** 1 – Use Eager Prefetch mode. 0 – Do not use Eager Prefetch mode (default).
27	Eager Prefetch for BAR1[1]	Eager Prefetch mode. On a Memory Read line or a Memory Read Multiple command by Memory Base Address Register 1 (BAR1), after the initial prefetch, if the Target Read continues without a disconnect and the Target Read FIFO has at least 8 data words empty, then prefetch the next block of data. **Value** / **Description** 1 – Use Eager Prefetch mode. 0 – Do not use Eager Prefetch mode (default).
26	MWMWI	Memory Write and Memory Write and Invalidate Behavior. In some system dependent applications, reducing the target write burst size on the Local Bus may help balance target read vs. write throughput. **Value** / **Description** 1 – Burst up to 8 words on the Local Bus. 0 – Burst up to 4 words on the Local Bus (default).

Table 12.16 PCI Target Control Register Field Descriptions (Part 1 of 4)

Notes

Bits	Field Name	Description
25:24	Threshold	Threshold for Target Write FIFO. The threshold setting provides hysteresis on the incoming PCI stream, ensuring that PCI burst writes are accepted in 4 or 8 data word increments. Note that the FIFO Threshold takes into account one command/address word FIFO location such that the actual internal FIFO pointer representation is the data word threshold + 1 command/address word. **Value** / **Description** 3 — Reserved. 2 — Threshold = 8 data words. Wait until at least 8 data words are free in FIFO before accepting any new write commands. 1 — Threshold == 4 data words. Wait until at least 4 data words are free in FIFO before accepting any new write commands (default). 0 — No Threshold (Threshold == 1 data word). Wait until at least 1 data word is free in FIFO before accepting any new write commands.
23	MRML4	Memory Read and Memory Line Behavior for Memory Base Address Register 4 (BAR4). **Value** / **Description** 1 — Prefetch 8 words on Memory Read and Memory Line target accesses similar to Memory Read Multiple. 0 — If the BAR for the decoded target Memory Read or Memory Read Line indicates the Prefetchable attribute is enabled, prefetch 4 words (default).
22	Reserved	Reserved to '0'.
21	MRML2	Memory Read and Memory Line Behavior for Memory Base Address Register 2 (BAR2). **Value** / **Description** 1 — Prefetch 8 words on Memory Read and Memory Line target accesses similar to Memory Read Multiple. 0 — If the BAR for the decoded target Memory Read or Memory Read Line indicates the Prefetchable attribute is enabled, prefetch 4 words (default).

Table 12.16 PCI Target Control Register Field Descriptions (Part 2 of 4)

Notes

Bits	Field Name	Description
20	MRML1	Memory Read and Memory Line Behavior for Memory Base Address Register 1 (BAR1). **Value** / **Description** 1 — Prefetch 8 words on Memory Read and Memory Line target accesses similar to Memory Read Multiple. 0 — If the BAR for the decoded target Memory Read or Memory Read Line indicates the Prefetchable attribute is enabled, prefetch 4 words (default).
19	EDT	Enable Discard Timer. When a master does not repeat a delayed read request within 2^{15} PCI clock cycles the PCI interface discards the delayed completion. When this bit is not set, delayed completions are never discarded. Note that an interrupt may optionally be set when the timer expires. **Value** / **Description** 1 — Discard timer enabled (default). 0 — Discard timer disabled.
18	RDR	Retry When Delayed Read. When this bit is set, all transactions are retried as long as there is an uncompleted delayed read being executed on the local bus. Once the PCI Target Read command is accepted, the PCI Target Write FIFO is flushed. Meanwhile, additional PCI Target Writes are accepted but not issued until after the PCI Target Read is issued. Once the PCI Target Write FIFO is flushed and the PCI Target Read is issued, all new PCI transactions are retried. **Warning**: setting this bit may violate the PCI 2.2 specification -- see implementation note in the PCI 2.2 specification Section 3.3.3.3.4. **Value** / **Description** 1 — Retry writes when delayed read. 0 — Post writes (default).

Table 12.16 PCI Target Control Register Field Descriptions (Part 3 of 4)

Notes

Bits	Field Name	Description
17:16	Reserved	Reserved to '0'.
15:8	DTimer	Disconnect Timer. This field specifies the number of PCI clock cycles the PCI interface will wait between data phases in an access before issuing a disconnect. A side effect of a disconnect is that any prefetched data in the Target Read FIFO will be flushed. The PCI 2.2 specification sets the maximum limit of this timer at 8 PCI clock cycles, but in some systems it may be necessary to extend this limit for more optimal performance. The minimum disconnect timer value is four. Values less than four are aliased to four. **Value** / **Description** any: PCI Target Interface will wait for subsequent data of an access before issuing a retry. 0x08: PCI Target Interface will wait 8 clocks for subsequent data of an access before issuing a retry (default).
7:0	RTimer	Retry Timer. This field specifies the number of PCI clock cycles the PCI interface will wait (to receive the first data of an access) before a retry command is issued. The PCI 2.2 specification sets the maximum limit of this timer at 16 PCI clock cycles, but in some systems it may be necessary to extend this limit for more optimal performance. **Value** / **Description** 0xFF-0x08: PCI Target Interface will wait the specified number of PCI clocks for the first data of an access before issuing a retry. 0x10: PCI Target Interface will wait 16 clocks for the first data of an access before issuing a retry. (default). 0x08: Minimum value allowed. 0x07-0x00: Reserved

Table 12.16 PCI Target Control Register Field Descriptions (Part 4 of 4)

1. To fully utilize the Eager Prefetch mode, the Target disconnect timer (DTimer) should be set high enough to avoid practically all disconnects, and the Master should issue Memory Read Multiple commands using large blocks (for instance 64 words or more).

PCI Arbitration Register

When the RC32334 PCI is in the host mode, either an internal arbiter or an external arbiter can be selected. The internal arbiter can arbitrate up to four[1] PCI masters, including the RC32334 device itself. When the internal arbiter is used, either a round robin or a fixed priority arbitration scheme can be chosen. If the RC32334 PCI is in the satellite mode, then the external arbiter is always used.

At boot time, in the standard reset boot mode, the PCI Target Not Ready Bit is set. This allows the PCI configuration registers to be written from the CPU and from the PCI side. After initialization, this bit should be cleared so that normal PCI operation, which requires the configuration registers to be in read-only mode, can begin.

Note: Most conventional masters are able to take advantage of Idle Grant Mode enabled.

1. Two PCI masters for the RC32332.

Notes

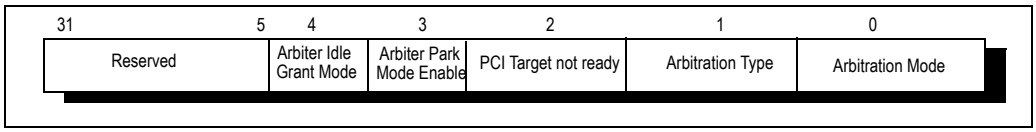


Figure 12.11 PCI Arbitration Register Fields

Bit	Field Name	Description
31:5	Reserved	
4	Arbiter Idle Grant Mode	Arbiter Idle Grant Mode. If enabled, the Arbiter Idle Grant Mode may use PCI Spec. rules to withdraw a Grant during Idle cycles and then Grant a higher priority Master. If disabled, the Arbiter will not withdraw a Grant unless the original requesting Master withdraws its request, the Master asserts pci_frame_n, or until a PCI Arbiter Timeout occurs where 16 PCI clocks occur after a Grant and pci_frame_n has not been asserted by the Master. **Value** / **Description** 1 – PCI Arbiter Idle Grant Mode Enabled. 0 – PCI Arbiter Idle Grant Mode Disabled (default).
3	Arbiter Park Mode Enable	Arbiter Park Mode Enabled. When the PCI bus interface is configured to operate in PCI host mode using an internal arbiter, this bit selects the bus parking mode to park the bus on the last master that was granted access to the bus. If the Arbiter Park Mode is disabled, then the PCI Arbiter will return to Idle when no masters are requesting the bus. **Value** / **Description** 1 – PCI Arbiter Park Mode Enabled. 0 – PCI Arbiter Park Mode Disabled (default).
2	PCI Target Not Ready	0 = PCI target ready (default if PCI-boot mode is selected) 1 = PCI target not ready (default if standard boot mode is selected)
1	Arbitration Type	0 = Use Internal Arbiter 1 = Use External Arbiter (default)
0	Arbitration Mode	0 = Round Robin. Rotating sequence is RC32334 PCI, pci_req_n[0], pci_req_n[1][1], pci_req_n[2], and so on. 1 = Fixed Priority (default). Priority order is RC32334 PCI, pci_req_n[0], pci_req_n[1][1], pci_req_n[2], with the highest priority assigned to the RC32334 PCI.

Table 12.17 PCI Arbitration Register Field Descriptions

1. There is no pci_req_n[1] in the RC32332.

PCI to CPU Memory/IO Space [1,2,3,4] Base Registers

Whenever local CPU memory is accessed via the PCI bus, the upper 4 bits of a PCI address are substituted to create a CPU physical address.

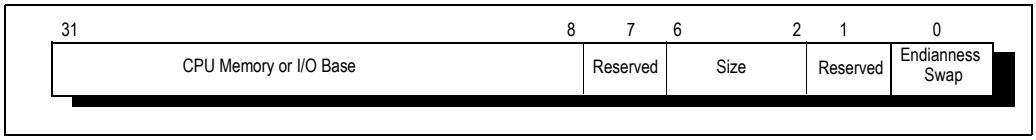


Figure 12.12 PCI to CPU Memory/IO Space [1,2,3,4] Base Register

Notes

Bit	Field Name	Description
31:8	CPU Memory or I/O Base	Default value is 0. Up to the top 24 bits translate/replace the top 24 bits of the PCI address with a Local Bus address. The Size Field determines the number of bits to translate/replace.
7	Reserved	Reserved to 0
6:2	Size	Address Space Size. This field indicates the size of the address space for the corresponding PCI base address register and the number of the CPU Memory or I/O Base bits to translate/replace. All bits greater than or equal to Size in the Memory or I/O Base Address Register (BAR) may be modified. Bits less than Size and greater than or equal to bit 4 always return a value of zero when read and cannot be modified. Setting the Size field to a value less than 8 disables the PCI base address register from decoding and causes the appropriate BAR bits to always return a value of zero when read and cannot be modified. One may also view this field as indicating the number of the Most Significant bit to be used in the decode, the minimum number being the 8th upper most significant bit. All bits greater than or equal to Size in the CPU Memory or I/O Base field are used to translate/replace the top bits of the PCI address with a Local Bus address. **Value** / **Description** 28 — BAR1: Set size to 2^{28} 27 — BAR1, BAR2, BAR4: Set size to 2^{27} 26 — BAR1, BAR2, BAR4: Set size to 2^{26} 25 — BAR2, BAR4: Set size to 2^{25} 1 — BAR2, BAR4: Set size 0, i.e., disabled (default for BAR2 and BAR4) 0 — BAR1: Set size to 2^{28} BAR3: Set size to 2^{8} (default for BAR1 and BAR3) Other — All other values are reserved. Summary of PCI BAR Size Decoding Valid Values BAR1: 28 (0), 27, 26 BAR2: 27, 26, 25, or disabled (1) BAR3: 8 (0) BAR4: 27, 26, 25, or disabled (1)
1	Reserved	Reserved to 0
0	Endianness swap	This bit controls byte swapping for PCI transactions that map to the local bus through the BAR register. **Value** / **Description** 1 — Byte swap 0 — No byte swap (default)

Table 12.18 PCI to CPU Memory/IO Space [1,2,3,4] Base Register Field Descriptions

Notes

PCI Configuration Address Register

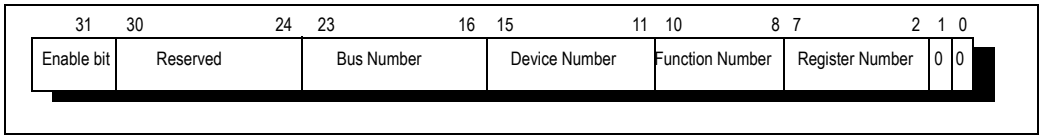


Figure 12.13 PCI Configuration Address Register Fields

Bit	Field Name	Description
31	Enable Bit	0 = Disabled[1] 1 = Enabled
30:24	Reserved	
23:16	Bus Number	PCI bus number
15:11	Device Number	PCI device number.[2] Asserts pci_ad[31:11] for device numbers 0x15 through 0x01.
10:8	Function Number	PCI function number
7:2	Register Number	PCI configuration register address
1:0		Hardwired to 00

Table 12.19 PCI Configuration Address Register Field Descriptions

1. If the enable bit is illegally disabled (because there is no analogous PC-AT I/O address space in the MIPS architecture) then the PCI target state machine is fully reset.

2. Device number 0 refers to the RC32334's host device (which is itself).

PCI Configuration Data Register

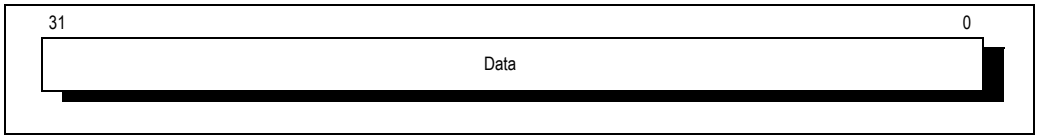


Figure 12.14 PCI Configuration Data Register Field

Bit	Field Name	Description
31:0	Data	Data value of configuration read/write access

Table 12.20 PCI Configuration Data Register Field Description

RC32334 PCI Configuration Registers

The PCI Configuration Space is described in this section. Table 12.21 shows the bits used, the read/write status, and the base address of each register. Shaded registers are read-only registers after being loaded and areas with x's are 'don't-cares'. Each of the registers is described in the sections following this table.

These shaded read-only registers can be written (where applicable and allowed) by the CPU by first enabling the PCI Target Not Ready bit in the PCI Arbitration Register and then following this two-step procedure:

1. Write the PCI Configuration Register Address as a pointer into the Register Number Field of the PCI Configuration Address Register.
2. Write the data to the PCI Configuration Data Register.

The non-shaded status and read/write registers in Table 12.21 may only be read or written by the CPU when the PCI Interface Controller is configured to be in Host Mode. When the PCI Interface Controller is configured to be in Satellite Mode, the non-shaded status and read/write registers may only be read by the

Notes

CPU by first enabling the PCI Target Not Ready bit in the PCI Arbitration Register and then following the two step pointer/data procedure listed in the paragraph above. In Satellite Mode, the non-shaded status and read/write registers may never be cleared or written by the CPU.

During a Configuration Register access or other access that results in an error—for example, an undecoded access to an empty PCI slot—the PCI controller will return the data value 0xFFFFFFFF to the CPU. A BusError will also result unless masked by the Internal BIU BusError Control bits for CPU and IP accesses. Typically, during empty slot polling, the Internal BIU BusError Control Register's bit 7 (BusError Exception Disable) can be disabled. This will prevent a CPU exception from being generated, and the BusError interrupt can be handled/ignored by the Expansion Interrupt Controller.

Bits Used 31 16 15 0				Address
Device ID		Vendor ID		00h
Status		Command		04h
Class Code			Revision ID	08h
BIST	Header Type	Master Latency Timer	Cacheline Size	0Ch
Memory/IO Base Address 1				10h
Memory/IO Base Address 2				14h
Memory/I/O Base Address 3				18h
Memory/IO Base Address 4				1Ch
xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	20h
xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	24h
xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	28h
Subsystem ID		Subsystem Vendor ID		2Ch
xxxxxxxx	xxxxxxxx	xxxxxxxx	xxxxxxxx	30h
Reserved				34h
Reserved				38h
Max_Lat	Min_Gnt	Interrupt Pin	Interrupt Line	3Ch
Reserved		Retry Timeout Value	TRDY Timeout Value	40h
Reserved				44h-FFh

Table 12.21 RC32334 PCI Configuration Registers

Vendor ID Register

This read/write register specifies the vendor of this device. This register must be set to 111Dh.

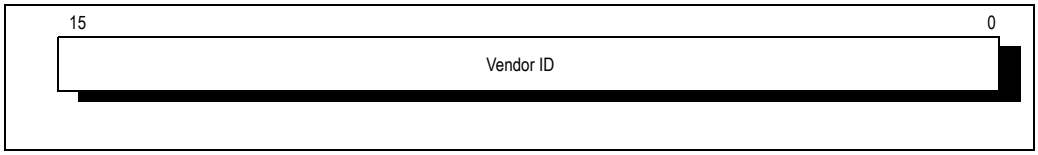


Figure 12.15 Vendor ID Register

Bit	Description	Reset
15:0	Vendor ID	111Dh

Table 12.22 Vendor ID Address Field Description

0x111D is the IDT Vendor ID.

Notes

Device ID Register

This read/write register specifies the ID to identify this device. On the RC32332 it is recommended that the PCI Device ID be written as 0205h, either through the configuration register interface or, if in the PCI Boot Mode, through the PCI Boot EEPROM. Using the recommended value will distinguish the controller from the RC32334. The default for the RC32332 is 204h, the same value as the RC32334.

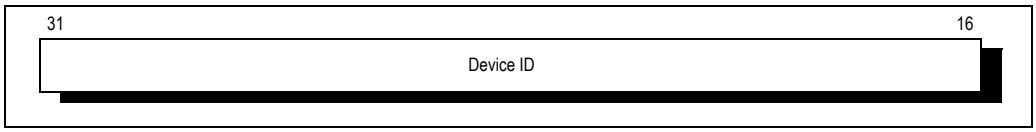


Figure 12.16 Device ID Register

Bit	Description	Reset
31:15	Device ID	0x0204h[1]

Table 12.23 Device ID Address Field Description

1. 0x0205h for the RC32332.

PCI Command Register

The PCI command register is a read/write register that provides protocol control to generate and respond to PCI cycles. Note that system errors typically occur due to address or data parity errors. However, if a target access occurs that is undecoded by the local memory bus, a system error will also occur, which is generally only recoverable by the master aborting its retries and resetting the RC32334.

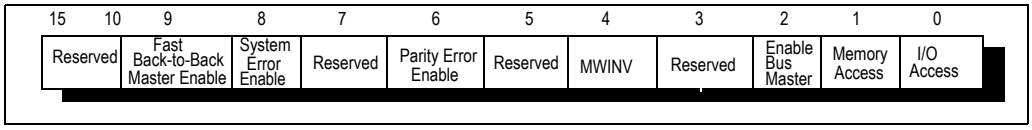


Figure 12.17 PCI Command Register

Bit	Description	Reset
15:10	Reserved	0h
9	Fast Back-to-Back Master Enable	0h
8	System Error Enable	0
7	Reserved	
6	Parity Error Enable	0
5	Reserved	
4	Memory Write and Invalidate Enable (MWINV). Allows Master write and invalidate operations if the Cacheline Size Configuration Register is nonzero and a burst write is issued from the DMA or CPU.	0
3	Reserved	
2	Bus Master Enable When PCI Boot mode option is selected at reset, PCI Master Enable Bit can be written to by the PCI Host because the PCI Target Not Ready bit (PCI Arbitration Register bit 2) is cleared when the PCI Boot mode option is selected at reset.	0
1	Memory Access Enable	0
0	I/O Access Enable	0

Table 12.24 Command Register

Notes

PCI Status Register

The PCI Status register reports the status of operations on the PCI bus. It also indicates the **PCI_DEVSEL#** timing that has been selected. If the Arbitration Register PCI Target Not Ready bit is set, then the 66MHz-Capable Flag (as well as other read-only flags) may be written.

Note: Except for bit 5, this is a register with a mixture of clearable status bits and read-only bits. Updates of bit 5 should be done through a read (other read-only bits), mask (status bits), modify, and write series of operations.

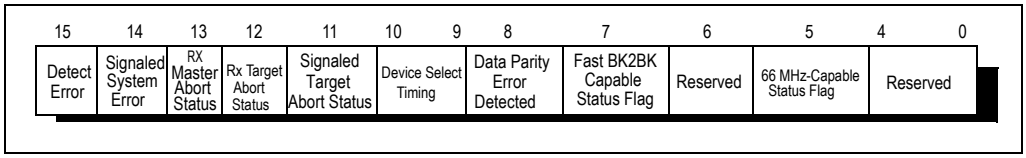


Figure 12.18 PCI Status Register

Bit	Description	Reset	Type
15	Detect Parity Error	0	Status
14	Signaled System Error	0	Status
13	Received Master Abort Status. Set when PCI Master terminates a Host-to-PCI transaction with a Master Abort.	0	Status
12	Received Target Abort Status. Set when the core initiates a PCI transaction and it is terminated by the Target.	0	Status
11	Signaled Target Abort Status	0	Status
10:9	Device Select Timing. Indicates timing of **PCI_DEVSEL#** when the core responds to a PCI transaction as a Target.	01	RO
8	Data Parity Detected	0	Status
7	Fast Back-to-Back Capable Status Flag.	1	RO
6	Reserved	0	RO
5	66 MHz-Capable Status Flag. Application software can set this bit to indicate the PCI interface can be operated at 66MHz.	1	Write/Read
4:0	Reserved	0h	RO

Table 12.25 Configuration PCI Status Register

Device Revision Identification Register

This read-only register contains the current revision identifier for this device.

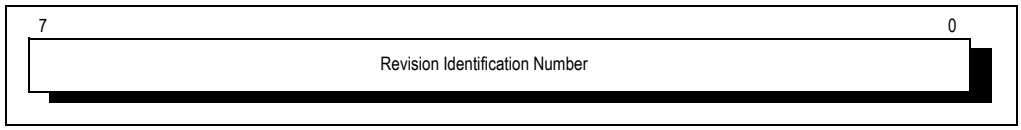


Figure 12.19 Configuration Device Revision Identification Register

Bit	Description	Reset
7:0	Revision Identification Number	01h

Table 12.26 Configuration Device Revision Identification Register Field Description

Notes

Class Code Register

The Class Code register contains a code value identifying the generic function of this device. The codes listed in Table 12.28 duplicate PCI 2.2 Specification.

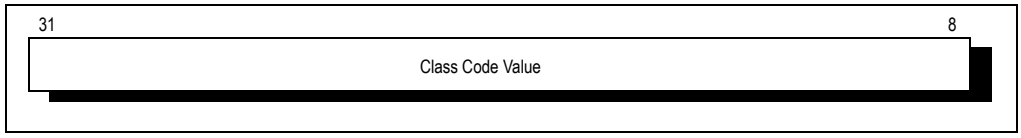


Figure 12.20 Class Code Register

Bit	Description	Reset
31:8	Class Code value	000000h

Table 12.27 Class Code Register Field Description

Base Class	Device type
00h	Device built before standardized definition of class codes
01h	Mass storage controller
02h	Network controller
03h	Display controller
04h	Multimedia device
05h	Memory controller
06h	Bridge device
07h	Simple communication controllers
08h	Base system peripherals
09h	Input devices
0Ah	Docking stations
0Bh	Processors
0Ch	Serial bus controllers
0Dh - FEh	Reserved
FFh	Device does not fit in any designated class

Table 12.28 Class Code Definitions

Cacheline Size

The cacheline size register specifies the system cacheline size in units of 32-bit words. It allows Master write and invalidate operations if the PCI Command Register MWINV bit is set and a burst write is issued from the DMA or CPU.

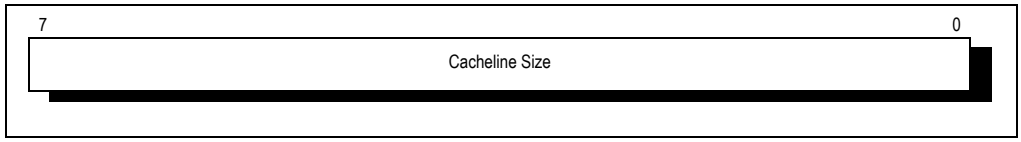


Figure 12.21 Cacheline Size Register

Notes

The PCI system requirements specify that the cacheline size must default to 0 at reset time. The RC32334 requires that the maximum cacheline size be no greater than 4.

Bit	Description	Reset
7:0	Cacheline Size	00h

Table 12.29 Configuration Cacheline Size Field Description

Master Latency Timer Register

The Master Latency Time Register is an 8-bit read/write register that controls the amount of time that the core, as a bus Master, can perform burst transfers if another Master requests the bus. The two least significant bits are hardwired to zero, allowing interval changes in increments of four clocks.

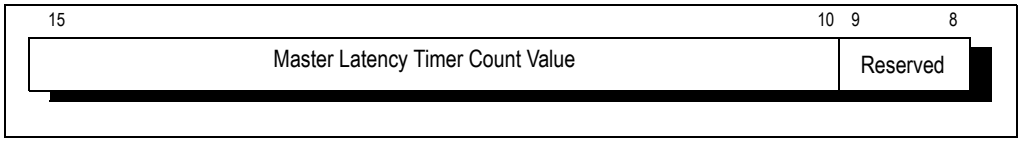


Figure 12.22 Master Latency Timer Register Fields

Bit	Description	Reset
15:10	Master Latency Timer Count Value This register sets the minimum number of PCI clock cycles that the core will be guaranteed access to the PCI bus. After the count has expired the core will surrender the PCI bus as soon as other PCI Master devices are granted the bus by the arbiter.	00h
9:8	Reserved: Hardwired to 0.	0h

Table 12.30 Master Latency Timer Register Field Descriptions

Header Type

Header Type is defined in Section 6.2.1 of the PCI 2.1 Specification.

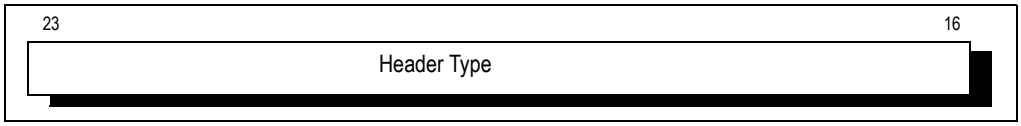


Figure 12.23 Header Type Register Field

Bit	Description	Reset
23:16	Header Type.	00h

Table 12.31 Header Type Register Field Description

BIST

This capability is not supported.

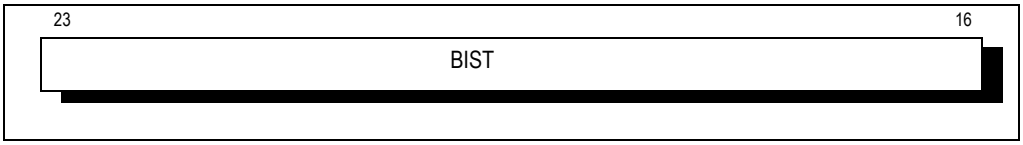


Figure 12.24 BIST Register Field

Notes

Bit	Description	Reset
23:16	Built in self test, hardwired to 0	00h

Table 12.32 BIST Register Field Description

PCI Memory/IO Base Address [1,2,3,4] Registers

This register contains the base address (BAR1-4) through which the PCI memory space is accessed. BARS1, 2, and 4 are memory base addresses by default; BAR3 is an I/O base address by default.

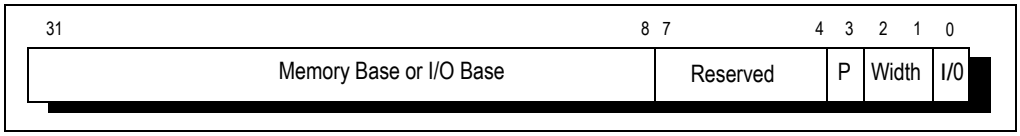


Figure 12.25 PCI Memory/IO Base Address [1,2,3,4] Register

Bits	Description	Reset
31:8	Memory or I/O Base Address	000000h
7:4	Reserved	0h
3	Prefetchable (hardwired to prefetchable). Value: 1 — Prefetchable (hardwired default) Value: 0 — Not prefetchable (reserved)	1
2:1	Port Bus Width. Hardwired to indicate 32 bit width.	00b
0	I/O Space vs. Memory Space (hardwired to Memory space). Value: 1 — I/O Space (reserved) Value: 0 — Memory Space (hardwired default)	0

Table 12.33 Memory/IO Base Address Register 1 (BAR1) Field Description

Notes

Bits	Description	Reset
31:8	Memory or I/O Base Address	000000h
7:4	Reserved	0h
3	Prefetchable. Note that if Not prefetchable is selected, then all types of PCI reads, including Memory Read, Memory Read Multiple, and Memory Read Line, will fetch 1 word at a time on the Local Bus to fulfill the exact number of words required for the read. **Value** / **Description** 1 – Prefetchable 0 – Not prefetchable (default)	0
2:1	Port Bus Width. Hardwired to indicate 32 bit width.	00b
0	I/O Space vs. Memory Space (hardwired to Memory Space) **Value** / **Description** 1 – I/O Space (reserved) 0 – Memory Space (hardwired default)	0

Table 12.34 Memory/I/O Base Address Registers 2 and 4 (BAR2,4) Field Description

.

Bits	Description	Reset
31:8	Memory or I/O Base Address	000000h
7:4	Reserved	0h
3	Prefetchable (hardwired to Not prefetchable) **Value** / **Description** 1 – Prefetchable (reserved) 0 – Not prefetchable (hardwired default)	0
2:1	Port Bus Width. Hardwired to indicate 32 bit width.	00b
0	I/O Space vs. Memory Space (hardwired to I/O Space) **Value** / **Description** 1 – I/O Space (hardwired default) 0 – Memory Space (reserved)	1

Table 12.35 Memory/I/O Base Address Register (BAR3) Field Description

Notes

Subsystem Vendor ID

This read/write register identifies the vendor of the subsystem where the PCI device resides.

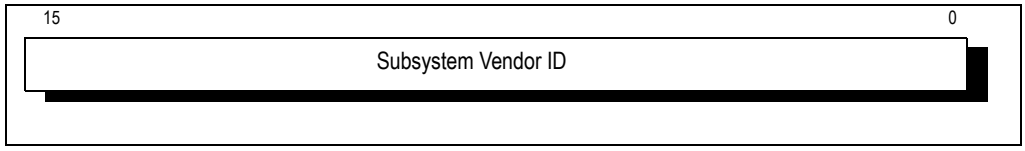


Figure 12.26 Subsystem Vendor ID Register

Bit	Description	Reset
15:0	Subsystem Vendor ID	0000h

Table 12.36 Subsystem Vendor ID Field Description

Subsystem ID

This read/write register identifies the subsystem where the PCI device resides.

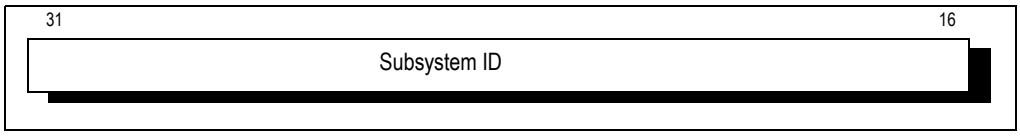


Figure 12.27 Subsystem ID Register

Bit	Description	Reset
31:16	Subsystem ID	0000h

Table 12.37 Subsystem ID Field Description

Interrupt Line Register

The interrupt line register contains the interrupt line to which the controller core is currently connected.

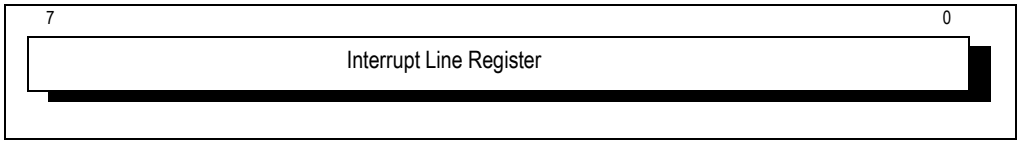


Figure 12.28 Interrupt Line Register

Bit	Description	Reset
7:0	Identifies the interrupt line register to which the core is connected	00h

Table 12.38 Interrupt Line Register Field Description

Interrupt Pin Register

This register contains the interrupt pin that the device uses.

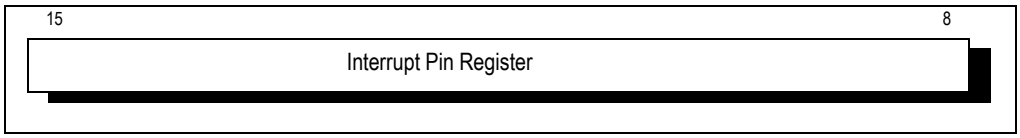


Figure 12.29 Interrupt Pin Register

Notes

Bit	Description	Reset
15:8	Identifies which interrupt pin the device uses	00h

Table 12.39 Interrupt Pin Register Field Description

MIN_GNT Register

This register specifies how long a burst period the device needs.

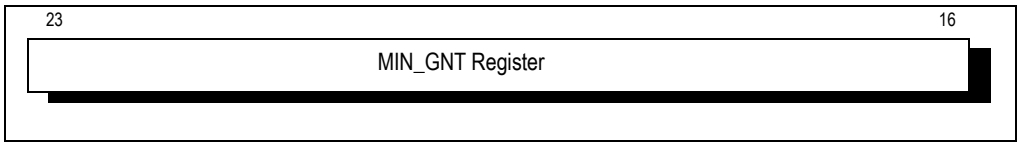


Figure 12.30 MIN_GNT Register

Bit	Description	Reset
23:16	Identifies length of burst period, assuming a 33 MHz clock. Units are 0.25 µS.	00h

Table 12.40 MIN_GNT Register Field Description

MAX_LAT Register

This register specifies how often the device needs to gain access to the PCI bus.

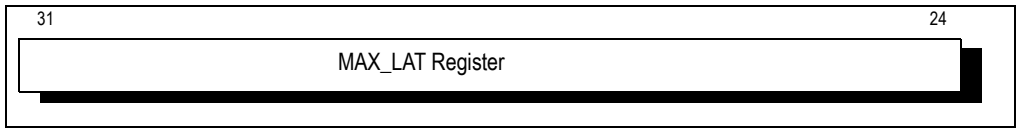


Figure 12.31 MAX_LAT Register

Bit	Description	Reset
31:24	Sets value of MAX_LAT. See PCI 2.1 specification Section 6.2.4 for details. Units are 0.25 µS.	00h

Table 12.41 MAX_LAT Register field Description

TRDY Timeout Value

This register sets the length of time in PCI clocks that the controller core, as master, will wait for TRDY.

Note: If this register is set to 0, the number of clocks that the Master waits for TRDY Timeout is infinite.

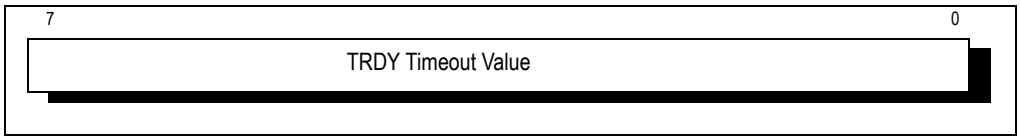


Figure 12.32 TRDY Timeout Value Register

Bit	Description	Reset
7:0	Sets number of PCI clocks that core as Master will wait for TRDY. The setting must be greater than or equal to 16. Settings of 15-0 are reserved.	80h

Table 12.42 TRDY Timeout Value Field Description

Notes

Retry Timeout Value

This register sets the maximum number of times the controller, as master, will retry. If the Retry Timeout Value is reached, a PCI Master Read or Write Error interrupt will occur in PCI Controller Interrupt Pending Register 11, bit 1 or bit 0.

The combined value (in nsec) of the Retry Timeout multiplied by TRDY Timeout must be smaller than the IPBus Timeout Value (in nsec). This ensures that PCI FIFO's are properly re-aligned on timeout errors. For additional information, refer to the BusError Address Register section in Chapter 8 and the Base Address Register 5 (Table 16.7) in Chapter 16.

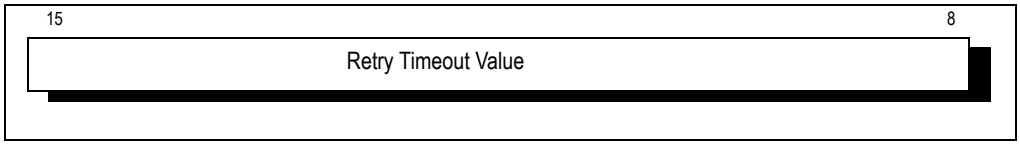


Figure 12.33 Retry Timeout Register

Bit	Description	Reset
15:8	Sets number of retries that the core as Master will perform.[1]	80h

Table 12.43 Retry Timeout Value Field Description

1. For example, assume 133MHz CPU, 33MHz PCI, and 1/2 system clock. If TRDY Timeout = 40 nsec and Retry Timeout = 40 nsec, then PCI Timeout = 124 µsec. and IPBus Timeouts should be greater than (124 µsec * 133MHz / 2) which should be greater than 2079h. Note that the CPU and IPBus Timers use the IPBus system clock which is typically 1/2 the frequency of the CPU pipeline clock.

For PCI systems capable of stopping the clock, the CPU Bus Timeout, IPBus Timeout, and the Watchdog Timeout timers must be disabled, so that the PCI clock can be restarted after an arbitrary delay. Alternatively, if the system design allows, the CPU could be signalled to not issue PCI Master transactions or a PCI reset could be issued when the PCI clock is stopped. Such a signal or reset would allow the CPU to continue operation while the PCI clock is stopped.

On PCI Master Write errors, the DMA engine is decoupled from the PCI interface via a master write FIFO. For example, if the PCI Master Write Error occurs due to a TRDY/Retry Timeout, the PCI Write FIFO is then flushed so that pending writes can be aborted. However, the DMA engine may have stored or continue to store additional writes after the initial error. Thus, in general, the PCI Master Write Error Interrupt service routine should note the PCI error and, if appropriate, restart the DMA engine from the point of the error.

DMA Controllers

Notes

Introduction

Four general purpose DMA channels move data between source and destination resources such as system memory, PCI or external I/O devices (8-,16-,or 32-bit I/O devices are treated as memory-mapped word-aligned devices). Using a flexible, memory-based descriptor structure, any of the four channels efficiently support “scatter/gather” capability. The RC32334 DMA supports byte, half-word (16-bit), word, and quad-word burst transfers that can cross over quad-word boundaries and are then automatically split into single-word transfers until a quad-word boundary is reached. The DMA controller also automatically prevents burst transfers from crossing SDRAM page boundaries and supports little- or big-endian data conversions.

To initiate[1] a DMA transfer, the CPU configures the Status, Source Address, Destination Address and Next Descriptor Address registers with the memory address, PCI bus address, read-write transfer direction, boundary crossing points, end-of-transfer interrupt enable, and transfer enable information. Once configured, the controller arbitrates for the memory and PCI bus and performs data transfers to or from memory without host CPU intervention.

Throughout this chapter, the following terms are used as defined below:

Transfer — refers to the cumulative data that is moved via the entire descriptor chain.

Transaction — pertains to data that is transferred per descriptor block.

List of Features

- *Four general purpose DMA channels*
- *Flexible descriptor based operation*
- *Memory-to-memory and memory-to-peripheral transfers*
- *Supports quad-word burst transfer*
- *Supports last partial word transfer*
- *Supports Endianness swapping*
- *Programmable DMA bus transaction burst size (1, 2, 4 or 16 bytes)*

DMA Enhancements in Y Silicon Revision

The DMA controller is one of the modules that has been enhanced in the Y revision of the silicon. Table 13.1 outlines some of the significant differences between the Z and Y revisions. For more information on the differences between the silicon revisions, refer to Application Note AN-350, RC32334/RC32332 Differences Between Z and Y Revisions, and to the RC32334/RC32332 Device Errata, both posted on IDT's web site at www.idt.com.

1.Although any of RC32334’s four DMA channels can be used for PCI master initiator reads or writes, channels 2 and 3 are recommended, because of the presence of the optional dma_ready_n pins for channels 0 and 1. Note that the RC32332 only includes the dma_ready_n signal for channel 0.

Notes

Function	Z Revision	Y Revision	Comments
Readability of DMA status registers	Not readable	Readable during active channel operation provided this function is enabled.	
Memory to PCI transfer behavior	Bus requested immediately when a memory controller has data to be transferred to the PCI module	Internal bus only requested once PCI master FIFO has enough space to accept transfer.	Reduces lock ups of the internal bus.
Priority of PCI DMA	Highest priority	Option to move this to lower priority level	Original user manual covering the Z revision incorrectly stated that PCI DMA was configured for lowest priority. Lowest priority level potentially useful for applications demanding high memory to memory performance but with relatively low PCI bandwidth requirements.

Table 13.1 DMA Differences Between Z and Y Revisions

Two new fields, shown in Table 13.2, have been added to the Configuration Register.

Bits	Field Name	Description
29	New Feature Mode	<table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>New Feature mode: Adds Status Register readability.</td></tr><tr><td>0</td><td>backward compatibility mode</td></tr></table>
20	SDRAM to PCI Arb Algorithm	SDRAM to PCI Arbitration Algorithm <table><tr><th>Value</th><th>Description</th></tr><tr><td>1</td><td>SDRAM to PCI write Arbitration waits for 4 words free or 1 word free in PCI Master TX FIFO depending on the burst size of the transfer.</td></tr><tr><td>0</td><td>backward compatibility mode (default).</td></tr></table>

Table 13.2 New Fields in DMA Configuration Register

Notes

Block Diagram

Functional units of the DMA device are shown in Figure 13.1.

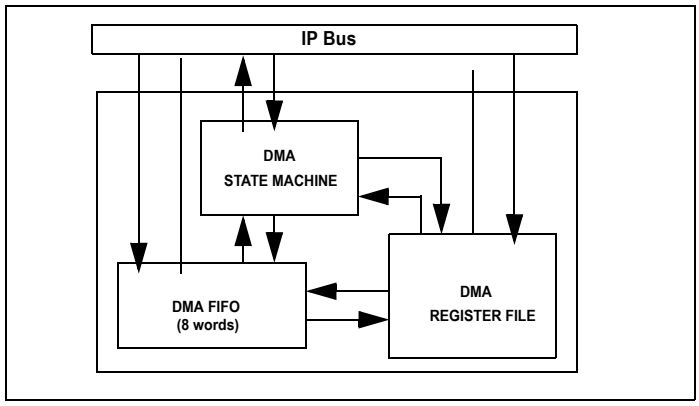


Figure 13.1 Diagram of DMA General Block with IP Bus Interface

DMA Operations

The RC32334 has four general purpose DMA channels to transfer data between memory, I/O and PCI. Channels 2 and 3 are recommended for use of PCI Initiated read/write. 8/16/32 bit I/O devices are treated as memory mapped word aligned devices.

The RC32334 DMA supports byte, half-word (16-bit), word, and quad-word burst transfer modes. Quad-word burst transfers that cross over quad-word boundaries are automatically split into single word transfers till a quad-word boundary is reached. The DMA automatically prevents burst transfers from crossing page boundaries.

The flexible descriptor structure allows the DMA controller to efficiently perform data transfers to or from memory without host CPU intervention. Each DMA channel has four registers to hold the current descriptor information. These are the status, source address, destination address and the next descriptor address registers. The functions of these registers are described later in the register definition section.

To begin a DMA transfer, the EnDMACh bit (bit 31 of the Configuration register) must first be set to 1 and the Base Descriptor Address register set to point to the first descriptor of a chain of descriptors in memory. The last descriptor in this chain is a dummy and as such is not affiliated with any valid data, but it is required to aid in terminating the DMA transfer. The status field of this dummy descriptor is set to 0, which sets the DMAOwn bit to 0.

The DMA loads the first descriptor from memory into its internal registers. The data transferred from the source device in to an internal FIFO and from this internal FIFO to the destination device. A DMA done interrupt can be generated. For this the user needs to set DMADnInt bit to 1, (i.e., bit 27 of the status register) for each transaction to tell the host that the current transaction has completed. The DMA will proceed to load the next descriptor via the address in the next descriptor address register to start a new transaction. This will continue until a dummy descriptor is detected in the chain.

The DMAOwn bit informs the DMA if the current descriptor is a valid data transaction descriptor. The LastDesc bit (bit 28 of the status register) informs the DMA if the current descriptor is the last valid data descriptor. If the DMA detects the dummy descriptor (i.e., when the DMAOwn bit is set to 0) the DMA will exit. However, if the DMA detects the dummy descriptor and does not detect the last valid data descriptor beforehand, then the DMA will still exit but will generate a DMA_not_owner interrupt via the external interrupt_n[2] pin. Please see Figure 13.2 for a detailed diagram of a DMA transfer configuration.

The RC32334 DMA internally supports byte swapping between little endian and big endian devices, which bridges the compatibility problems between two systems with different endianness. The RC32334 DMA supports last partial word accesses in any mode by generating the appropriate byte enable signal for the last partial data. In this way, a transfer of any byte length can occur in any DMA mode.

Notes

Two DMA modes (ready and done) are available for data transfer when accessing slow I/O devices. In the ready mode, when a slow I/O device is ready, the slow device asserts the dma_ready_n pin (low active) to initiate the data transfer. In the done mode, when a slow I/O device is done, the slow device asserts the dma_ready_n pin to signal to the DMA that the slow device is done receiving the current data. In both modes, the slow I/O device can keep the dma_ready_n pin de-asserted (high) if the slow device is not ready. This holds the DMA engine in the current state and does not start a new data transfer. Only DMA channels 0 and 1 have the dma_ready_n pins, so only these channels may be used to transfer data to or from slow I/O devices.

The DMA module includes a option (enabled via bit 20 in the configuration register) to allow SDRAM to PCI DMA transfers to occur without locking up the internal bus. When enabled, the bus is only requested once the PCI Master TX FIFO has enough space to accept the transfer. This feature is specifically designed to improve SDRAM to PCI transfers. However the read operation may occur from any type of memory, including that located on the local bus. Note that when this mode of operation is selected, the user must set up descriptors for that channel such that all destination addresses point to the memory space mapped to the PCI Master TX FIFO. If incorrectly configured, unnecessary delays may occur, since the arbitration for that channel will always check the PCI Master TX FIFO status, without checking whether the destination address resides in the PCI Master TX FIFO range or (for example) SDRAM memory. Thus, for example, if the destination is SDRAM memory, the PCI TX FIFO may be full due to an independent access from the CPU or other DMA channel. This would delay the SDRAM access, even though it is independent from the PCI Master TX transaction.

Endianness Swapping

The RC32334 DMA internally supports byte swapping between little-endian and big-endian devices, which bridges the compatibility issues that occur between two systems with different endianness. The RC32334 DMA supports last partial word accesses in any mode by generating the appropriate byte enable signal for the last partial data. In this way, a transfer of any byte length can occur in any DMA mode. The user must program the SrcEnd bit (Source Endianness) and the DstEnd bit (Destination Endianness). Examples of endianness swapping for word or half-word transfers are shown below.

	31...24	23...16	15...8	7...0
Big Endian	A	B	C	D
Little Endian	D	C	B	A

	31.....16		15.....0	
Big Endian	B	A	D	C
Little Endian	D	C	B	A

DMA Transfer Modes

For word/burst transfers (32<->32), the starting address must be word aligned. However, the RC32334 DMA will complete unaligned transfers also (32<->32) by automatically converting to the byte transfer mode, independent of the word or the quad-word burst transfer settings.

1. **Byte transfers**: used for non-word aligned transfers
   - *Program MaxburstSz = 000, 1 byte.*
   - *Both source and destination devices can have any starting address. Depending on the address and endianness, the data will show up on the proper byte lanes.*
   - *For each transfer, the DMA will request the bus, read one byte from the source into internal fifo, write that byte from internal fifo to the destination, then release the bus. The DMA will repeat this procedure until all the data are transferred.*

Notes

2. **Half-word transfers**: typically used for data that are represented as a 16-bit integer
   - *Program MaxBurstSz = 001, 2 bytes.*
   - *Both source and destination starting addresses must be half-word aligned. Depending on the address and endianness (must be 32 bits wide), the data will show up in the proper byte lanes.*
   - *For each transfer, the DMA will request the bus, read one half-word from the source into an internal fifo, write that half-word from this internal fifo to the destination, then release the bus. The DMA will repeat this procedure until all the data are transferred.*
   - *If the last data is one byte only, the DMA will generate the appropriate byte enable signal for that byte.*
3. **Word transfers** with word-aligned starting address
   - *Program MaxburstSz = 010, 4 bytes.*
   - *Both source and destination have word-aligned starting addresses.*
   - *For each transfer, the DMA will request the bus, read one word from the source into internal fifo, write that word from this internal fifo to the destination, then release the bus. The DMA repeats this procedure until all the data are transferred.*
   - *If the last data is a partial word, the DMA will generate the appropriate byte enable signal for that partial word.*
4. **Quad-word burst transfers** with word-aligned starting address
   - *Program MaxburstSz = 100, 16 bytes.*
   - *Both source and destination starting addresses are word-aligned and not decremented.*
   - *For burst transfer, the DMA will request the bus, read four words from the source into an internal fifo, write four words from this internal fifo to the destination, then release the bus. The DMA will repeat this procedure until all the data are transferred.*
   - *If either the source or the destination starting address is not at a quad-word boundary, then the DMA will read or write single words until the address reaches a quad-word boundary. Then the DMA will start quad-word burst transfers.*
   - *If the last data is a partial word, the DMA will generate the appropriate byte enable signal for that partial word.*
5. **Unaligned word/burst transfers**
   - *The DMA will automatically detect unaligned transfers and ignore the user-programmed MaxburstSz and degrade the DMA transfer into the byte mode transfer.*
   - *For each transaction, the DMA will request the bus, read one byte from the source into an internal fifo, write that byte from this internal fifo to the destination, then release the bus. The DMA will repeat this procedure until the data are transferred.*

DMA Transfer Operations

A DMA sequence begins after the DMA channel has been enabled via its Configuration register. The DMA engine begins by accessing the Base Descriptor Register to locate the physical address pointer for the first descriptor, which consists of 4 words and is usually located in a large memory buffer. Each descriptor has four fields. Fields one and two describe the Status and Source Address that are to be read. Fields three and four describe the Destination Address that are to be written and the Next Descriptor Pointer.

Descriptors are often located one after another; however, because each descriptor contains a full 32-bit pointer in the Next Descriptor Address register, they can be located anywhere in memory on a quad-word aligned boundary. The Next Descriptor Address register points to the memory location from which the next descriptor is to be fetched. The DMA engine then uses the Base Descriptor to fetch (read) the first descriptor. Thus, a 4-word burst read of the descriptor will occur.

Notes

Note that the Bus Turnaround on the physical system data bus after a descriptor fetch is hardcoded to 1.0 clock. Thus, descriptor memory tables should be set up in physical memory that uses 1.0 clock BTA or less.

Once the first descriptor is fetched, the DMA engine arbitrates for the System Data bus; when granted, the DMA engine executes a read to the Source Address as pointed to by the descriptor. A write is then immediately issued to the Destination Address, as pointed to by the descriptor. The DMA engine continues to request/grant/read/write until its Status indicates that the transaction is complete. When the DMA engine completes the request/grant/read/write loop for a descriptor, then the Status is written back to the Status Word within the descriptor's 4-word memory location. If more descriptors are pending, then the DMA engine uses the Next Descriptor Address register to fetch the next descriptor from memory, via a burst 4-word read. A diagram of the DMA transfer configuration details is provided in Figure 13.2.

The DMA engine is often instructed by the descriptor chains to endlessly loop through the descriptor pool. One exception is to run out of descriptors "owned" by the Controller; for example, to run out of memory buffers. In this case, after fetching the next descriptor, the DMA engine examines the previous "LastDesc" status bit and ends/disables if "LastDesc" is set to"1." This method is also the typical way to end a fixed-size chain of descriptors, such that the DMA engine fetches one extra dummy descriptor with the DMA Own status bit set to the CPU.

A second exception is to fetch a Controller owned descriptor with the "LastDesc" status bit set to '0', to indicate an unexpected last condition such that a "LastDesc" interrupt is generated. In this case, DMA may be restarted/re-enabled with the "continuation" control set by the Interrupt Handler when a descriptor becomes available, such that a new descriptor is re-fetched from the Next Descriptor Address register. This optional restart feature allows software to maintain the DMA channels with a Base Descriptor Address as a constant, if desired.

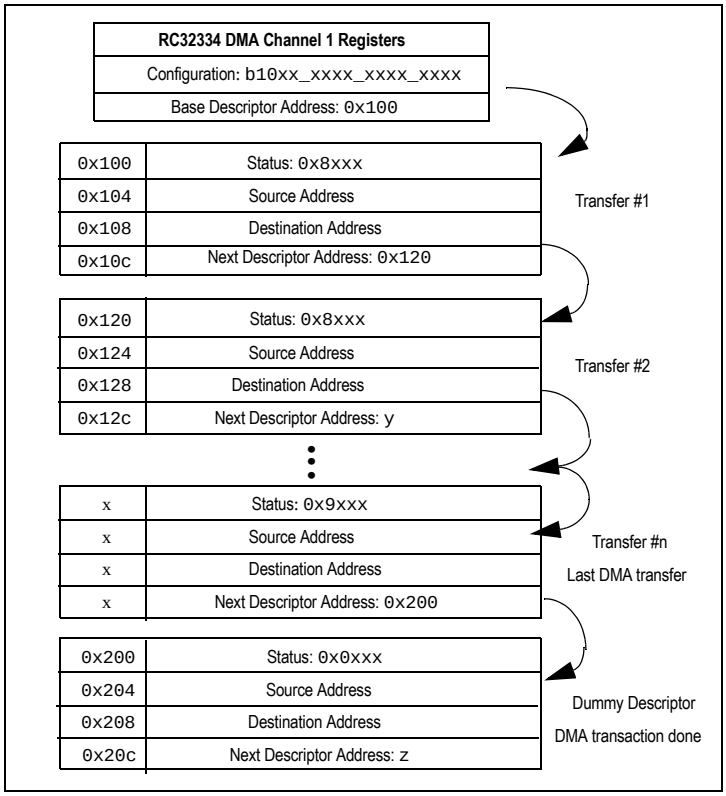


Figure 13.2 DMA Transfer Configuration

Notes

Last Partial Word Transfers

For **word or burst transfers**, if the last data is a partial word (for example, 1, 2 or 3 bytes), the DMA will always read the data from the low address of the source and write it to the low address of the destination, if the address is incremented (or high address if the address is decremented). For example, if the last transfer is one byte and the address is incremented, the last byte will show up on the following byte lane:

	31...24	23...16	15...8	7...0
Big Endian	**A**			
Little Endian				**A**

For **half-word transfers**, if the last data is a byte, the DMA will always read the last byte from the low address (in little endian order) of the source and write it to the low address (in little endian order) of the destination. For example, if A1 is 0, the last byte will show up on the following byte lane:

	31...24	23...16	15...8	7...0
Big Endian		**A**		
Little Endian				**A**

If A1 is 1, the last byte will show up on the following byte lane:

	31...24	23...16	15...8	7...0
Big Endian				**A**
Little Endian		**A**		

Transfer Restrictions

When implementing DMA operations, the following transfer restrictions must be considered:

- *When the source or destination address is a constant (such as in I/O devices), the address must be word-aligned, and the I/O devices must be connected to the appropriate byte lanes according to endianness*
- *The following transfers are not supported:*
  - *(1) Source is incremented and destination is decremented*
  - *(2) Source is decremented and destination is incremented*
- *Unaligned word/burst transfers can only be done in byte mode (user-programmed MaxburstSz is ignored for unaligned word/burst transfer)*
- *When an address is decremented or constant, the DMA will not support burst transfers*
- *The starting address must be half-word aligned for half-word transfers*
- *Devices must have the same port width when doing DMA transfers from I/O to I/O*
- *DMA channels 2 and 3 do not have the dma_ready_n pins, therefore they can not be used to do DMA transfers with slow I/O devices.*

DMA Arbitration Methods

The RC32334's four independent DMA channels are functionally identical—with the exception of priority coding—and initialized with a set of chaining registers to determine the DMA source start base address, the DMA target start base address, the data transfer number, and the protocol style selection.

As discussed earlier in the transfer operations section of this chapter, to begin a data transfer operation, the channel will first arbitrate for the System Data bus. With multiple DMA requests pending, after a DMA access, the System Data bus is granted to the Controller instead of the next highest requestor; as such, there are two priority tiers: bus requestors and Controller. If DMA receives the bus, the DMA will use either the fixed or rotating priority schemes.[1] The rotating arbitration scheme is illustrated in Figure 13.3.

Notes

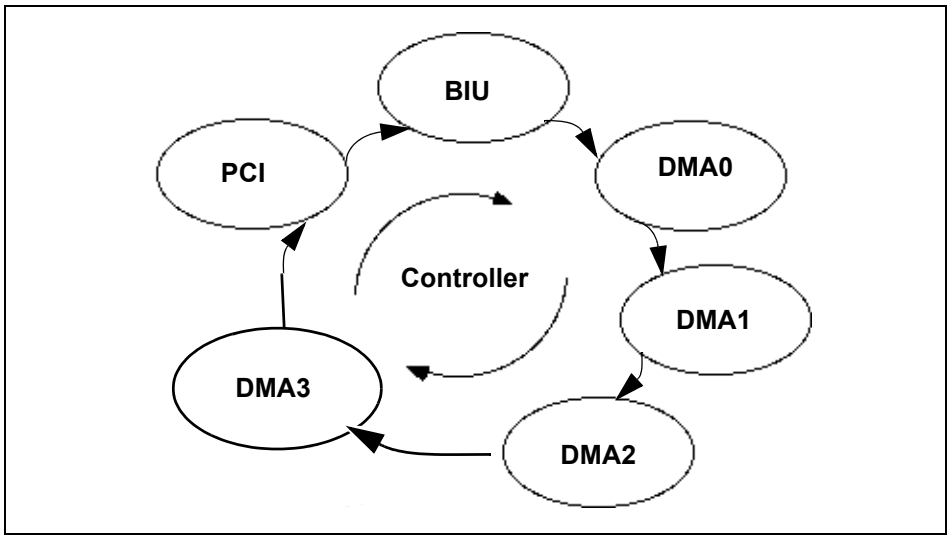


Figure 13.3 Diagram Showing the Rotating Arbitration Scheme

The fixed priority encoding scheme is illustrated in Table 13.3.

Fixed Priority	Agent
Highest	BIU
•	PCI
•	DMA0
•	DMA1
•	DMA2
Lowest	DMA3

Table 13.3 Fixed Priority Encoding

Once arbitration is settled, the DMA channel generates a read cycle with the source base address. The control register determines whether it is a burst. Typically, the source address will be through an internal memory controller, which will take the address and generate data, acknowledges, etc., back to the DMA controller channel. The DMA controller uses the DMA 4-word deep buffer FIFO to absorb the potential burst read data.

After the read is completed, the DMA channel initiates a write to the target address by emptying the read buffer FIFO. As in the read, the write is typically through an internal memory controller on the I/O Controller. This internal memory controller takes the address and data from the DMA FIFO and generates a write transaction. At the end of the transaction, the DMA channel's Block Size register is decremented by the transaction length.

If the Block Size register has not reached 0, the source and target addresses are incremented to their next value (which could be by +0, +1, +2, +4, or +16, depending on whether incrementing is enabled and whether a mini-burst or burst occurred). If the Block Size register has reached 0, then the DMA channel is finished with its current descriptor link chaining register assignment and the Status Word is written back to the memory descriptor. If the control register so instructs, the channel may set an interrupt and/or stop, and/or it may reload a new descriptor of chaining registers. If a new descriptor is loaded, then the DMA channel will repeat the basic DMA channel transaction by copying the new descriptor's instructions into the current instructions and then executing them.

1.DRAM refreshes occur in the background and may override an access to DRAM by delaying the start of the access.

Notes

Bus Turnaround (BTA) clock cycles will only be inserted if the DMA write after a read is going to take less than the BTA value programmed. See bus interface unit register descriptions for more information.

DMA Access

On a DMA access that results in an IPBus Error, such as to a non-existent PCI target, the BIU Arbiter behavior is changed to more gracefully generate an IPBus Timeout rather than a Watchdog Timeout.

Signal Definitions

Two modes are available to the user for completing data transfers to or from slow I/O devices: dma_done_n and dma_ready_n[1].

DMA Ready

The RC32334 DMA Controller has a DMA throttling option called DMA Ready. The DMA Ready option is typically used for one of several cases:

- *External read I/O device where overall data rate is much slower than CPU, such that occasional reads are done in the process background on a request demand basis*
- *External write I/O device where overall data rate is much slower than CPU, such that occasional reads are done in the process background on a request demand basis*
- *External read I/O FIFO device that has FIFO almost Full Flag*
- *External write I/O FIFO device that has FIFO almost Full Flag.*

The dma_ready_n input signal can be used by an external I/O device to demand that the RC32334 DMA Controller initiate one transfer of data to or from the I/O device. dma_ready_n[0] can be used to control DMA Channel 0, and dma_ready_n[1] can be used to control DMA Channel 1.

Note: On the RC32332, there is one flow control signal, dma_ready_n[0] for DMA Channel 0.

dma_ready_n is first sampled 1.0 clock after the fourth/last debug_cpu_ack_n asserts from the DMA Channel Descriptor fetch. This is similar to Figure 13.4, except that the transaction is a four-word burst read. dma_ready_n can be a 1.0 clock pulse, or it can be asserted longer, as long as it is de-asserted (if it is intended to be de-asserted) before the next dma_ready_n sampling point for the next DMA transfer. If dma_ready is kept asserted at this sampling point, then the DMA Controller will assume that the next transfer is to occur (as might be the case if the external I/O device is a FIFO device that keeps dma_ready_n asserted until empty).

As shown in Figure 13.4, the next dma_ready_n sampling point first occurs on the clock after write debug_cpu_ack_n occurs for the current DMA transaction write. Note that if the transaction has multiple data, then the sampling point is after the last data. Whether or not the external I/O device is reading or writing (source or destination), the sampling always occurs after the read portion of the DMA transaction and the write portion have both occurred. After the sampling point occurs, if dma_ready_n is not asserted, then the DMA Controller will pause on that channel until dma_ready_n is asserted. Finally, after being sampled, dma_ready_n is internally double registered. Thus the next DMA transaction cannot possibly occur until at least 2.0 clocks after dma_ready_n is asserted.

1.As noted under DMA restrictions, only DMA channels 0 and 1 have the dma_ready_n pin, making these two channels the only DMA channels available for this type of data transfer. Channels 2 and 3 are restricted from this type of DMA transfer operation. The RC32332 only includes the dma_ready_n pin for DMA channel 0.

Notes

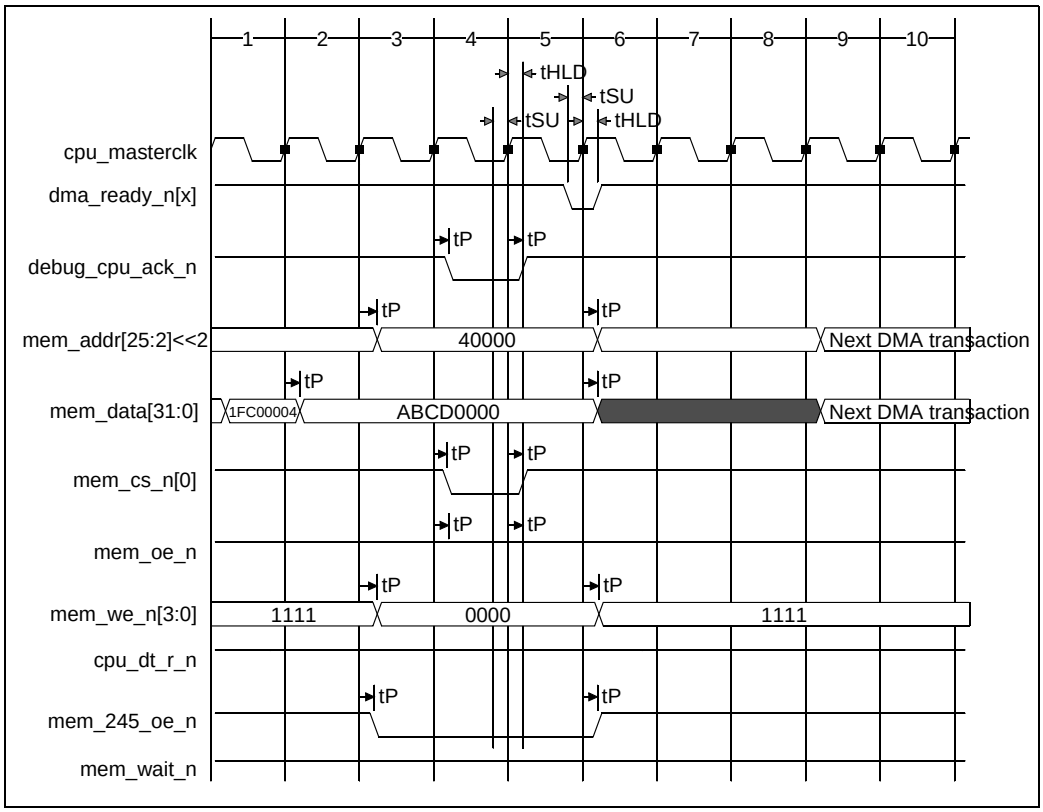


Figure 13.4 DMA Ready Sampling Point

Figure 13.4 shows the end of a DMA transaction, where the read portion has already occurred and the write portion is occurring via the 32-bit Memory Controller location. Note that the SDRAM Controller, as well as the other modes of the Memory Controller, have similar cases relative to the final assertion of debug_cpu_ack_n. The first possible point at which dma_ready_n is sampled to initiate another DMA transaction occurs 1.0 clock after debug_cpu_ack_n occurs.

Note that if the Enable DMA Channel bit in the channel Configuration Register is disabled during a burst transfer where dma_ready_n is being used, the burst transfer may abort the writeback of words from the DMA FIFO if the write is not block aligned. If the Enable DMA Channel is to be used (to disable the channel) in conjunction with the dma_ready_n mode, the transfer address should be aligned with the MaxBurstSz so that the final writes are flushed to memory.

DMA Done

DMA Done mode uses the dma_ready_n pin. The DMA channel configuration register bit 27, DMADone bit turns this mode on or off. Block devices which initially request or send more data than may be necessary can benefit from the use of the DMA Done mode.

DMA Pin	Type	Function
dma_ready_n[1:0]	Input	**dma_ready mode:** Input pins for DMA channels 0 and 1 to indicate that the I/O device is ready for the next data in the current DMA descriptor transaction
dma_done_n[1:0]	Input	**dma_done mode:** Input pins for DMA channels 0 and 1 to indicate that the I/O device is finished with the current descriptor transaction

Table 13.4 DMA Signal Pins and Definitions

Notes

The DMA Done mode allows the dma_done_n pin to abort and disable the DMA channel either:

- *at the end of the current DMA bus transaction or*
- *at the end of the next DMA bus transaction.*

DMA channel interrupt #3 is asserted and an Interrupt Service Routine can setup and re-enable the DMA channel as desired.

The dma_done_n pin is required to be asserted for at least one clock cycle. It is internally synchronized by double clocking to help avoid metastability issues if asserted asynchronously. As shown in Figure 13.5, at the end of clock state cycle #11, dma_done_n is sampled by each DMA bus transaction exactly 5.0 clocks previous to the final internal cpu_ack_n signal which can be seen on the RC32334 as the debug_cpu_ack_n signal. Both single data and burst accesses sample the dma_done_n signal 5.0 clocks previous to the final debug_cpu_ack_n of the read/writeback phase. For example, in a 4 word burst DMA bus transaction, the final debug_cpu_ack_n is the 8th ack, which occurs after the 4 word DMA read ack's, on the 4th word of the DMA writeback.

If the dma_done_n pin is asserted after the dma_done_n internal sample point, then the DMA channel will cease after the next DMA bus transaction from this DMA channel. The next DMA bus transaction address cycle which can be started from the same DMA channel will take at least 11.0 clocks from the previous dma_done_n internal sample point.

The DMA channel interrupt #3 for the dma_done_n pin occurs concurrently with the final debug_cpu_ack_n assertion and will occur whether or not the DMA channel coincidently ends because of the descriptor finishing normally. Note that interrupt #3 is different from the DMA Done Interrupt as setup by the descriptor status field which typically is used to denote the end of a normal descriptor finish.

After the interrupt for the dma_done_n pin occurs, an Interrupt Service Routine can setup or reuse a new descriptor and restart the DMA channel by re-enabling it.

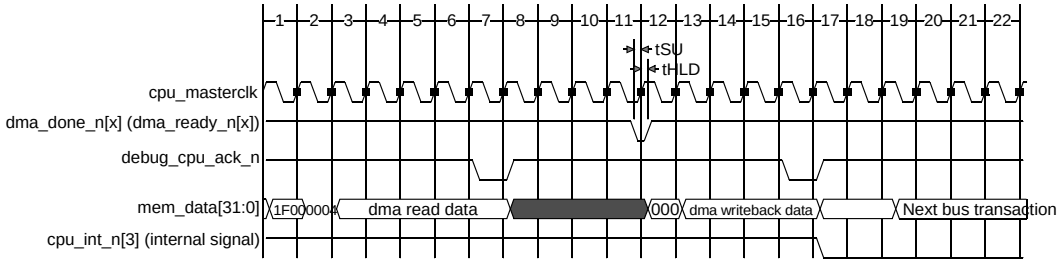


Figure 13.5 DMA Done Timing Diagram

Internal DMA Interrupt Signals

Each of the four DMA channels has 3 interrupts that are routed to the Expansion Interrupt Controller, which provides the logic for software to analyze the various interrupts generated by the overall system. These internal interrupts perform the functions described in Table 13.5. More details on the Expansion Interrupt Controller are provided in Chapter 14 of this manual.

Internal DMA Interrupt Pin	Type	Function
interrupt_n[0]	Output	DMADnInt, DMA Done interrupt, which is generated at the end of each descriptor frame if the descriptor status register has the DMADnInt interrupt bit 27 enabled.
interrupt_n[1]	Output	Dma_end_early interrupt, which is caused by bus error or timeout.
interrupt_n[2]	Output	Dma_not_owner interrupt, which is caused by the following situation: DMAOwn = 0 for current descriptor frame and LastDesc = 0 for the previous descriptor frame.

Table 13.5 DMA Interrupt Definitions (Part 1 of 2)

Notes

Internal DMA Interrupt Pin	Type	Function
interrupt_n[3]	Output	Dma Done pin asserted when in the dma_ready_n pin's Dma_done mode and dma_done_n is asserted and once the current DMA transaction completes. Still asserted even if the channel completes simultaneously or if the channel is disabled.
interrupt_n[4]	Output	Dma Halt asserts DMA interrupt bit 4 when the DMA channel completes all descriptor frames, including the flushing of the FIFO of a final transfer and the dummy descriptor fetch. This interrupt is also asserted when the Enable DMA-Channel bit in the DMA Channel Configuration Register is cleared while the channel is active and allows software to monitor when the channel has completed its current transfer.

Table 13.5 DMA Interrupt Definitions (Part 2 of 2)

Restarting DMA Channels

DMA channels are restarted by re-enabling the EnDMACh field in the DMA Channel Configuration Register after this field has been cleared by the DMA Engine. An idle DMA channel can be detected using one of the following methods:

1. Use the DMA_clr_en interrupt. After the interrupt occurs, clear the interrupt and re-enable the DMA channel. After the interrupt occurs, clear the interrupt and re-enable the DMA channel.
2. Program DMA Done Interrupt to occur on the Last Descriptor. Then clear the interrupt and wait until the dummy descriptor is fetched before re-enabling the DMA channel. Usually, the interrupt handler has sufficient delay for the dummy descriptor to be fetched. Alternatively, the DMA channel's Configuration Register (or any other DMA channel register) can be polled (if any of the fields beside the Configuration Register's EnDMACh field were initially set to non-zero) because the registers return zeroes until the dummy descriptor fetch is finished. If the DMA Configuration Register's New Feature field is set, the EnDMACh field (in the Configuration Register) itself can be polled to detect if the dummy descriptor fetch is finished.
3. Program the Last Descriptor to not have the LastDesc field set. The dummy descriptor will be fetched and DMA_not_owner interrupt will occur. From context, if the descriptors are not added dynamically, or if the interrupt does not occur immediately after a descriptor is added dynamically, the interrupt can be cleared, and the DMA channel can be re-enabled using the Configuration Register's EnDMACh field. Note that, if applicable, in the interrupt after a dynamic descriptor case, either of the following may occur:
   - *Update (if necessary) the Next Descriptor Register and set the Configuration Register Cont field*
   - *Update the Base Descriptor Register and set the Configuration Register Cont field.*

   Then, the interrupt can be cleared and the DMA channel can be re-enabled using the Configuration Register's EnDMACh field.

Register Mapping and Descriptions

Each of the four DMA channel's control registers determines channel usage, data transfer modes, and descriptor ownership of the four independent, general purpose channels. As programmed, these channels move data between source and destination ports, such as system memory, PCI, or external I/O devices. For the address mapping tables listed in this section, the effective address for a specific set of registers for that channel is the Base Address plus the Offset, as indicated in the tables that follow.

Notes

Base Address Channel 0	Register Name	Offset Address	Effective Address Channel 0
1800_1400	Configuration Register	00	Base + Offset
	Base Descriptor Register	04	
	Current Address Register	08	
	Status/Block Size Register	10	
	Source Address Register	14	
	Destination Address Register	18	
	Next Descriptor Address Register	1C	

Table 13.6 DMA Channel 0 Register Address Map

Base Address Channel 1	Register Name	Offset Address	Effective Address Channel 1
1800_1440	Configuration Register	00	Base + Offset
	Base Descriptor Register	04	
	Current Address Register	08	
	Status/Block Size Register	10	
	Source Address Register	14	
	Destination Address Register	18	
	Next Descriptor Address Register	1C	

Table 13.7 DMA Channel 1 Register Address Map

Base Address Channel 2	Register Name	Offset Address	Effective Address Channel 2
1800_1900	Configuration Register	00	Base + Offset
	Base Descriptor Register	04	
	Current Address Register	08	
	Status/Block Size Register	10	
	Source Address Register	14	
	Destination Address Register	18	
	Next Descriptor Address Register	1C	

Table 13.8 DMA Channel 2 Register Address Map

Notes

Base Address Channel 3	Register Name	Offset Address	Effective Address Channel 3
1800_1940	Configuration Register	00	Base + Offset
	Base Descriptor Register	04	
	Current Address Register	08	
	Status/Block Size Register	10	
	Source Address Register	14	
	Destination Address Register	18	
	Next Descriptor Address Register	1C	

Table 13.9 DMA Channel 3 Register Address Map

Configuration Register

The Configuration Register is a 32-bit register containing the data used to implement and manage DMA controller functions, as shown in Table 13.10. When set to 1, bit 31 of this register is used to enable a DMA channel after the descriptor information and address have been established. Once initiated, DMA transfers can only be disabled after the current transaction is complete. When the DMA channel is active, this register reads as 0.

On DMA channels 0 and 1, the DMA Done (bit 27) and DMA Ready (bit 28) modes are available for data transfers from slow I/O devices. In the ready mode, once a slow I/O device is ready to begin a data transfer, the I/O device will set the dma_ready_n pin, which initiates the transfer. In the done mode, the slow device asserts the dma_ready_n pin to signal the DMA that the slow device is done receiving the data.

In both modes the slow device can hold the dma_ready_n pin high if the device is not ready, which holds the DMA engine in the current state and a new data transfer is not initiated. Note that if the DMA transfers are to be modulated using the dma_ready_n pin, with the DMA enable bit in the configuration register disabled, burst transfers should be word (4 bytes) sized. Figure 13.6 illustrates the fields of the Configuration register and Table 13.10 provides the description and initial value of those fields.

The DMA status registers are readable during active channel operation whenever the New Feature mode is turned on.

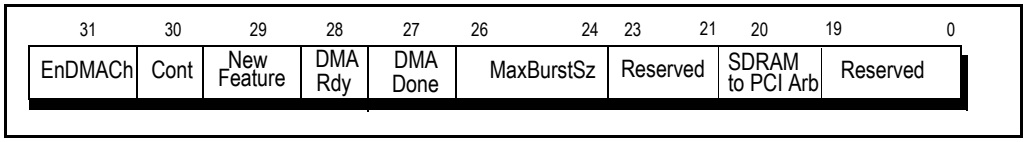


Figure 13.6 Configuration Register Fields

Bits	Field Name	Description	Initial Value
31	EnDMACh	Enable DMA Channel Used to enable a DMA channel after descriptor information and address is set up. **Note:** This bit is not normally cleared (DMA disabled) by the user after a DMA transfer has started. Usually, the DMA engine itself will clear this bit after the transaction has been completed. **Value** / **Description** 1 — Enable DMA channel 0 — DMA channel is idle	0

Table 13.10 Configuration Register Field Descriptions (Part 1 of 3)

Notes

Bits	Field Name	Description	Initial Value
30	Cont	Continuation After a DMA transfer failure, this bit specifies which descriptor to restart the transfer from. **Value** / **Description** 1 – Restart from failed descriptor 0 – Restart from the first descriptor	0
29	New Feature Mode	**Value** / **Description** 1 – New Feature Mode: Adds Status Register readability. 0 – Backward compatibility mode.	0
28	DMARdy	DMA Ready External Pin Wait for dma_ready_n input (in ready mode). Wait for the dma_ready_n input before the next bus arbitration. DO NOT assert bits 28 and 27 simultaneously, since it will lead to undefined behavior. **Value** / **Description** 1 – Enable dma_ready_n pin (in ready mode) 0 – Ignore dma_ready_pin (in ready mode)	0
27	DMADone	DMA Done External Pin Wait for dma_ready_n input (in done mode). Put the dma_ready_n pin into dma_done_n mode. If asserted, dma_done_n will stop the DMA Channel immediately after the current DMA bus transaction completes and disable the DMA Channel (also see EnDMACh bit 31). If the DMA Channel is waiting for bus mastership, then asserting dma_done_n will stop the DMA Channel immediately after the next DMA bus transaction completes and disable the DMA Channel. DO NOT assert bits 28 and 27 simultaneously, since it will lead to undefined behavior **Value** / **Description** 1 – Enable dma_ready_n pin (in done mode) 0 – Ignore dma_ready_pin (in done mode)	0

Table 13.10 Configuration Register Field Descriptions (Part 2 of 3)

Notes

<table>
<tr><th>Bits</th><th>Field Name</th><th>Description</th><th>Initial Value</th></tr>
<tr><td>26:24</td><td>MaxBurstSz</td><td>Maximum Burst Transaction Size Bit Field
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>111</td><td>Reserved</td></tr>
<tr><td>110</td><td>Reserved</td></tr>
<tr><td>101</td><td>Reserved</td></tr>
<tr><td>100</td><td>16 bytes</td></tr>
<tr><td>011</td><td>Reserved</td></tr>
<tr><td>010</td><td>4 bytes</td></tr>
<tr><td>001</td><td>2 bytes</td></tr>
<tr><td>000</td><td>1 byte</td></tr>
</table></td><td>000</td></tr>
<tr><td>23:21</td><td>Reserved</td><td></td><td>0</td></tr>
<tr><td>20</td><td>SDRAM to PCI Arb Algorithm</td><td>SDRAM to PCI Arbitration Algorithm
Note: For PCI master write transactions only, this field should be enabled. For all other types of transactions, such as memory to memory transactions or PCI master read transactions, this field should be disabled.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>1</td><td>SDRAM to PCI write Arbitration waits for 4 words free or 1 word free in PCI Master TX FIFO depending on the burst size of the transfer.</td></tr>
<tr><td>0</td><td>Backward compatibility mode (default).</td></tr>
</table></td><td></td></tr>
<tr><td>19:0</td><td>Reserved</td><td></td><td>0</td></tr>
</table>

Table 13.10 Configuration Register Field Descriptions (Part 3 of 3)

Base Descriptor Address Register

This 32-bit register is used in conjunction with bit 31 of the Configuration Register, to initiate a DMA transfer. However, before a DMA transaction can begin, the Base Descriptor Address register must be set to point to the physical address of the first descriptor in a chain of descriptors in memory. The base address does not change after the transfer has started. When the DMA channel is active, this register reads as 0.

Fields of the Base Descriptor Address register are shown in Figure 13.7. Fields of this register are described in Table 13.11.

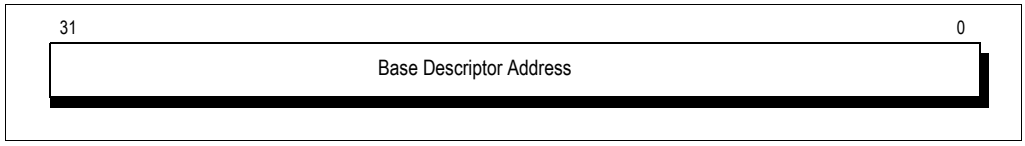


Figure 13.7 Base Descriptor Address Register Field

Notes

Bits	Field Name	Description
31:0	Base Descriptor Address	Used in conjunction with bit 31 of the Configuration register, this Quad-word boundary physical address pointer points to the first descriptor of a chain of descriptors in memory

Table 13.11 Base Descriptor Address Field Description

DMA Example

```
/*
        Use DMA Channel 0 to read/write 3 words, 1 word at a time to/from
        SRAM Memory buffer A, to/from SRAM Memory buffer B, and from
        buffer B to SRAM Memory buffer C.
*/
/* algorithm:
   1. SETUP DMA COMMAND REGISTERS
   2. INITIALIZE DESCRIPTORS
   2.1. INITIALIZE DESCRIPTORS
   2.2. INITIALIZE DUMMY DESCRIPTOR
   3. SETUP INTERRUPT CONTROLLER
   4. ENABLE DMA
   5. WAIT FOR DMA DONE INTERRUPT
*/

  /* 1. SETUP DMA COMMAND REGISTERS */
     $display($stime,": CPU 32bit 1wd write to DMA Config Reg");
     //       address     data            be
     cpu_wr_1wd('h1800_1400, `DWIDTH'h0000_0000, `BWIDTH'h0);

     $display($stime,": CPU 32bit 1wd write to DMA Base Desc Reg");
     //       address     data            be
     cpu_wr_1wd('h1800_1404, `DWIDTH'h1fc0_0200, `BWIDTH'h0);

  /* 2.1. INITIALIZE DESCRIPTOR 0 */
     $display($stime,": CPU 32bit 1wd write to Mem0: Desc.0.0 (Status)");
     //       address     data            be
     // DMAOwn bit31 needs to be set to DMA, no DMADnInt bit27
     // Read/Write 1 wd at a time for 3 wds
     cpu_wr_1wd('h1fc0_0200, `DWIDTH'h8000_000c, `BWIDTH'h0);

     $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.1 (Source)");
     //       address     data            be
     cpu_wr_1wd('h1fc0_0204, `DWIDTH'h1fc0_0100, `BWIDTH'h0);

     $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.2 (Dest)");
     //       address     data            be
     cpu_wr_1wd('h1fc0_0208, `DWIDTH'h1fc0_0600, `BWIDTH'h0);
     $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.3 (Next)");
```

Notes

```
        //      address     data            be
        cpu_wr_1wd('h1fc0_020C, `DWIDTH'h1fc0_0210, `BWIDTH'h0);

    /* 2.2. INITIALIZE DESCRIPTOR 1:LAST */
        $display($stime,": CPU 32bit 1wd write to Mem0: Desc.1.0 (Status)");
        //      address     data            be
        // DMAOwn bit31 needs to be set to DMA, DMADnInt bit27
        // DMA LastDesc bit 28 needs to be set
        // DMA DnInt bit 27 optionally set
        // Read/Write 1 wd at a time for 3 wds
        cpu_wr_1wd('h1fc0_0210, `DWIDTH'h9800_000c, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.1.1 (Source)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_0214, `DWIDTH'h1fc0_0600, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.1.2 (Dest)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_0218, `DWIDTH'h1fc0_1000, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.1.3 (Next)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_021C, `DWIDTH'h1fc0_0260, `BWIDTH'h0);

    /* 2.3. INITIALIZE DUMMY DESCRIPTOR */
        $display($stime,": CPU 32bit 1wd write to Mem0: Desc.0.0 (Status)");
        //      address     data            be
        // DMAOwn bit31 needs to be set to CPU
        cpu_wr_1wd('h1fc0_0260, `DWIDTH'h0000_0000, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.1 (Source)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_0264, `DWIDTH'hffff_ffff, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.2 (Dest)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_0268, `DWIDTH'hffff_ffff, `BWIDTH'h0);

        $display($stime,": CPU 32bit 1wd wr to Mem0: Descriptor.0.3 (Next)");
        //      address     data            be
        cpu_wr_1wd('h1fc0_026C, `DWIDTH'hffff_ffff, `BWIDTH'h0);

    /* 3. SETUP INTERRUPT CONTROLLER */
        $display($stime,": CPU 32bit 1wd write to INT Clear Reg");
        //      address     data            be
        cpu_wr_1wd('h1800_0578, `DWIDTH'hffff_ffff, `BWIDTH'h0);
        #(10.0*`CLK_PERIOD+000);    // align for next transaction;
```

Notes

```
$display($stime,": CPU 32bit 1wd write to INT Mask Reg");
//      address     data            be
cpu_wr_1wd('h1800_0574, `DWIDTH'h0000_0001, `BWIDTH'h0);

/* 4. ENABLE DMA */
$display($stime,": CPU 32bit 1wd write to DMA Config Reg");
//      address     data            be
// 1 wd at a time
cpu_wr_1wd('h1800_1400, `DWIDTH'h8200_0000, `BWIDTH'h0);
```

Current Address Register

This 32-bit register is managed by the DMA and contains the physical address of the descriptor in memory associated with the current transaction. When the DMA channel is active, this register reads as 0.

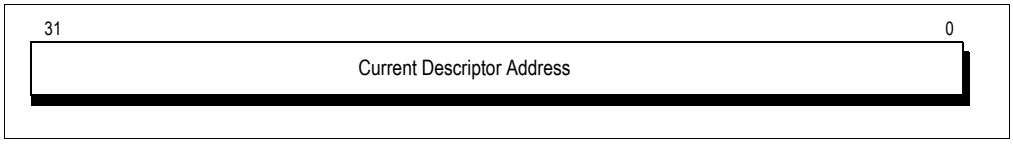


Figure 13.8 Next Descriptor Address Field

Bits	Field Name	Description
31:0	Current Descriptor Address	Quad-word boundary physical address pointer points to the current descriptor in the chain of descriptors

Table 13.12 Current Descriptor Address Field Description

Source Address Register

This 32-bit register contains the physical address of the memory location from which the next data is to be read. Figure 13.9 and Table 13.13 describe and illustrate the fields of this register. This register is internally updated after each DMA read transfer. However, the register can only be accessed by software when the DMA channel is idle. When the DMA channel is active, this register reads as 0.

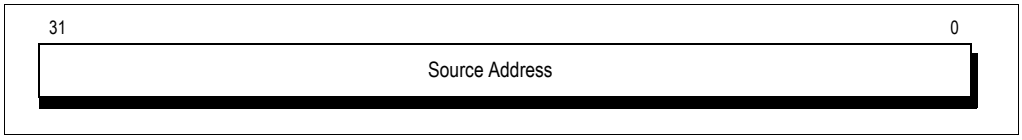


Figure 13.9 Source Address Field

Bits	Field Name	Description
31:0	Source Address	Points to the next physical address to be read

Table 13.13 Source Address Register Field Description

Destination Address Register

This 32-bit register contains the physical address of the memory location where data is to be written. Fields of the Destination register are illustrated and described in Figure 13.10 and Table 13.14. This register is internally updated after each DMA read transfer. However, the register can only be accessed by software when the DMA channel is idle. When the DMA channel is active, this register reads as 0.

Notes

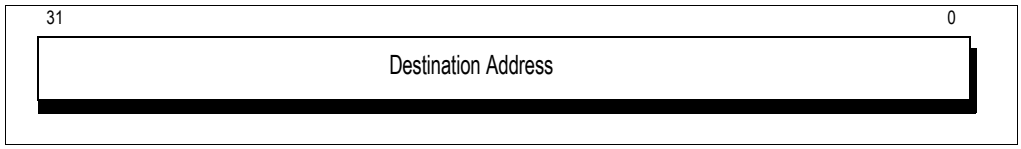


Figure 13.10 Destination Address Fields

Bits	Field Name	Description
31:0	Destination Address	Points to the next physical address to be written to

Table 13.14 Destination Address Field Description

Next Descriptor Address Register

This 32-bit register contains the physical address of the descriptor in memory that is next in line to be operated on. This register is illustrated and described in Figure 13.11 and Table 13.15. When the DMA channel is active, this register reads as 0.

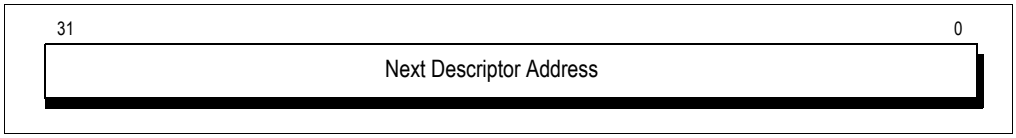


Figure 13.11 Next Descriptor Address Field

Bits	Field Name	Description
31:0	Next Descriptor Address	Quad-word boundary physical address pointer points to the next descriptor in the chain of descriptors

Table 13.15 Next Descriptor Address Field Description

Status Register

Each descriptor of 4 words contains a status register that is subdivided into 11 fields, which are illustrated in Figure 13.12 and described in Table 13.16. This register is internally updated after each DMA read transfer. However, the register can only be accessed by software when the DMA channel is idle. When the DMA channel is active, this register reads as 0.

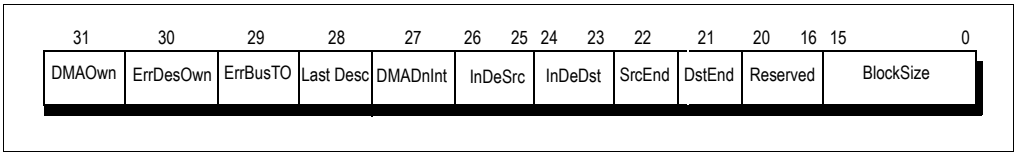


Figure 13.12 Status Register Fields

Bits	Field Name	Description	Initial Value
31	DMAOwn	DMA is Owner Value 1: DMA controller owns the current descriptor Value 0: The processor owns the current descriptor	0

Table 13.16 Status Register (Part 1 of 3)

Notes

Bits	Field Name	Description	Initial Value
30	ErrDes-Own	**Error On Descriptor Ownership** (descriptor under flow) If DMA does not own the current descriptor and the previous descriptor is not the last one, then this error bit is set in this register, but it is not written back to the descriptor in memory. 1 - Error on the descriptor ownership 0 - No Error	0
29	ErrBusTO	Error due to Bus Error or Timeout If a bus error or timeout occurs during a descriptor fetch, during a DMA transaction, or during a descriptor write back, then this error bit is set. This register is not written back to the descriptor in memory. Note that errors (for example, to undecoded memory space) during reads are always reported, but errors during writes may not be reported, especially in cases where the write occurs to a write buffer (FIFO). For example, PCI Master Write errors may not necessarily report an error to DMA, but will report a PCI Interface Master Abort error. **Value** / **Description** 1 – Bus error or bus timeout 0 – No error	0
28	LastDesc	Last valid data descriptor This field indicates whether or not the descriptor is the last descriptor in the list of descriptors. If LastDesc is set and the next descriptor's DMAOwn bit is not set, the DMA channel terminates normally. If LastDesc is not set and the next descriptor's DMAOwn bit is not set, the ErrDesOwn Descriptor Ownership Error status bit will be set. If the next descriptor's DMAOwn bit is set, the LastDesc bit is ignored. **Value** / **Description** 1 – Is the last descriptor 0 – Is not the last descriptor	0
27	DMADnInt	Assert DMA Done Interrupt This field generates an interrupt after all the transactions associated with the current descriptor are done. **Value** / **Description** 1 – Generate interrupt 0 – Do not generate interrupt	0
26:25	InDeSrc	Increment/Decrement Source Memory **Value** / **Description** 11 – Reserved 10 – Constant 01 – Decrement 00 – Increment	0

Table 13.16 Status Register (Part 2 of 3)

Notes

Bits	Field Name	Description	Initial Value
24:23	InDeDst	Increment/Decrement Destination Memory Value / Description: 11 Reserved; 10 Constant; 01 Decrement; 00 Increment	00
22	SrcEnd	Source Endianness This field specifies endianness of the source/data/part/device. PCI DMA has endianness controlled via the PCI Bridge and will always swap PCI from Little-endian to Big-endian. Value / Description: 1 Big Endian; 0 Little Endian	0
21	DstEnd	Destination Endianness This field specifies endianness of the destination part/device. PCI DMA has endianness controlled via the PCI Bridge. Value / Description: 1 Big Endian; 0 Little Endian	0
20:16	Reserved		0
15:0	BlockSize	Block Size This field specifies the number of bytes to be transferred in the current transaction descriptor. This field is internally updated after each DMA write transfer. However, the field can only be accessed by software when the DMA channel is idle. When the DMA channel is active, this field reads as 0. The value '0' is reserved and should not ordinarily be written to this field/descriptor if the descriptor is intended for an actual transfer. The value '0' will transfer 4 bytes rather than a zero length transfer of 0 bytes.	0

Table 13.16 Status Register (Part 3 of 3)

Timing Diagrams

Figure 13.13 illustrates an entire DMA transaction transferring 2 words of memory to another memory location. The transaction begins with a bus request/grant assertion. Then a 4-word burst read of the 1st descriptor is fetched. The bus request/grant is de-asserted. At this time, another DMA channel or the Controller could take over the system. Note that the Bus Turnaround after the descriptor fetch is hardcoded to 1.0 clock.

Another bus request/grant assertion occurs. Then the DMA source read occurs, in this case 1 word. After the DMA read, Bus Turnaround idle cycles occur. Then the DMA destination write occurs, in this case 1 word. Another bus request/grant de-assertion occurs. At this time, another DMA channel or the Controller could take over the system bus. In cases were the length of the transaction and the burst size allow, a burst read and burst write may occur at this step.

Another bus request/grant assertion occurs. The DMA read/write pair repeats.

Notes

Another bus request/grant assertion occurs. In this case, the length of the DMA transfer is only 2 words, so the DMA is complete. If more words were to be transferred, DMA would continue to loop through the DMA read/write sequence until the transfer was complete. Since DMA has completed, a status writeback of 1 word occurs, writing 1 word back to the descriptor. Another bus request/grant de-assertion occurs. At this time, another DMA channel or the Controller could take over the system.

Another bus request/grant assertion occurs. Even if the current descriptor is the last valid descriptor, one more “dummy” descriptor burst read fetch occurs in order to support descriptor memory buffer underflow algorithms. Another bus request/grant e-assertion occurs. At this time, another DMA channel or the Controller could take over the system bus.

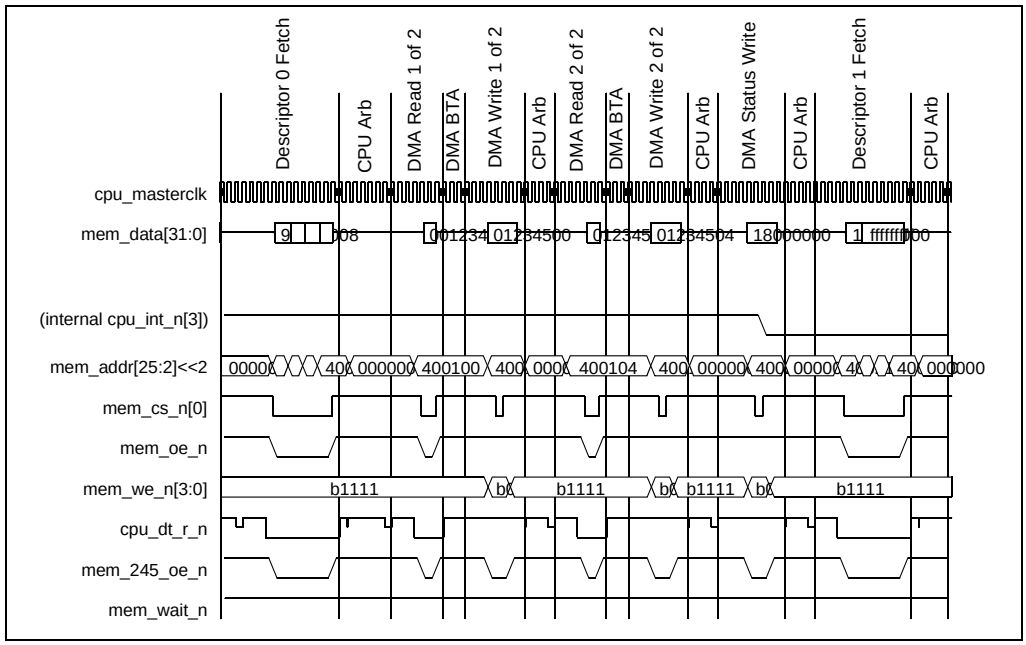


Figure 13.13 Two Word SRAM to SRAM Access by DMA

Notes

Expansion Interrupt Controller

Notes

Introduction

The RC32334 Expansion Interrupt Controller provides the logic for software to process the overall system interrupts generated by the RC32334, and it adds to the circuitry and control already provided by the RC32300's CPU Core Coprocessor 0 (CP0) registers. The RC32334 Expansion Interrupt Controller registers each system interrupt and provides the pending status, which can be used to automatically generate a hardware interrupt to the CPU core via the individual mask bits for each interrupt. These mask bits enable software to allow/disallow each individual interrupt and to propagate or not propagate to the overall interrupt.

The pending interrupt status can also be optionally set or cleared by a direct software write. Also, for software convenience, a masked write to the Interrupt Clear register allows a per bit clearing of pending interrupts. In addition to the CPU interrupt generation, a dedicated register for PCI interrupt generation is provided. The same software interface is provided so that interrupts are steered to generate a hardware interrupt on the pci_inta_n (pci_gnt_n[2]) pin, when PCI is in the satellite mode.

Features

- *Combines all interrupts into a single CPU interrupt*
- *Combines all CPU- to- PCI mailbox interrupts into a single PCI interrupt*
- *Pending Register Bit for each interrupt*
- *Mask Register Bit for each interrupt*
- *Software Clear Register for clear per bit writes*

Block Diagram

The Expansion Interrupt Controller diagram is shown in Figure 14.1 and the Group/Bit-Slice diagram is shown in Figure 14.2

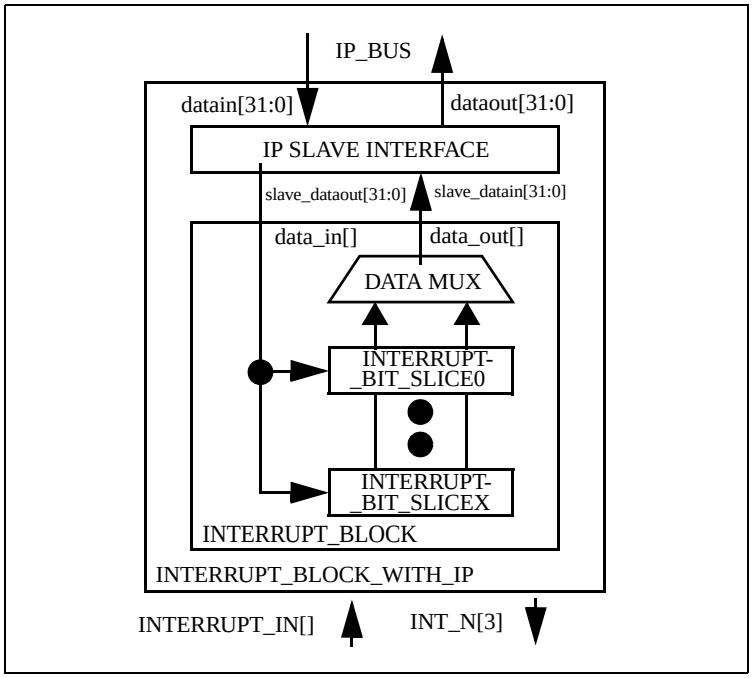


Figure 14.1 Expansion Interrupt Controller Block Diagram

Notes

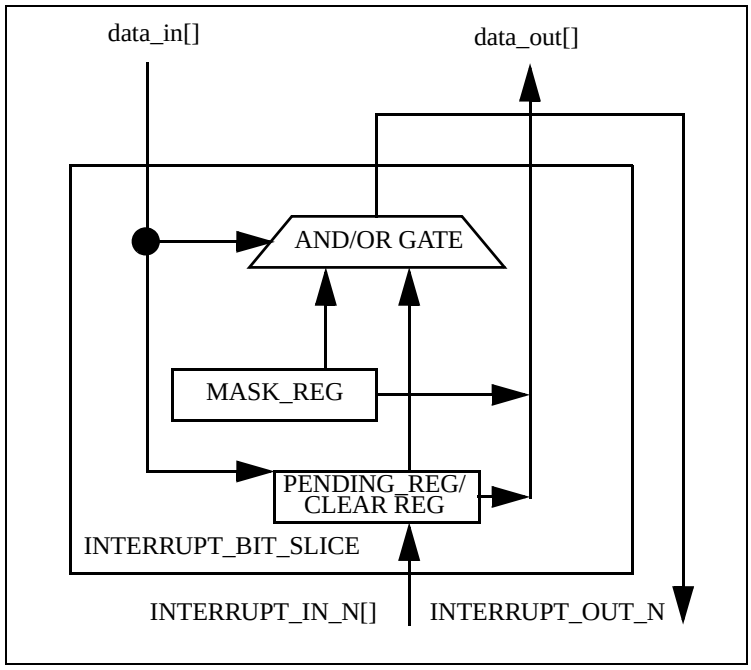


Figure 14.2 Expansion Interrupt Block Diagram Group/Bit-Slice

Operational Overview

The Expansion Interrupt Controller extends the RC32300's CPU Core CP0 interrupt control by collating the RC32334 generated interrupts into a single CPU interrupt. When a general purpose interrupt is received, the Interrupt Service Routine (ISR) first saves CPU registers, checks its Cause Register and then checks its Pending Interrupt Register. If the pending interrupt is from the on-chip peripheral modules, then the ISR checks the Expansion Interrupt Controller Pending Interrupt Register. After treating/noting the interrupt condition, the ISR resets the pending interrupt by writing to the corresponding bit in the Expansion Interrupt Clear Register. The ISR can then exit by restoring the CPU registers and executing an RFE instruction.

Interrupts can be independently masked by the Expansion Interrupt Mask Register. When an ISR is first called, the general RC32334 CPU CP0 Global Interrupt Enable bit is disabled. The ISR can then implement priority interrupts by first changing the RC32334 Expansion Interrupt Mask bits accordingly and re-enabling the RC32334 CPU CP0 Global Interrupt Enable bit.

Device specific interrupt conditions are discussed in the chapter appropriate for the device. For more information on bus errors and their causes, see Chapter 8, RC32334 Internal Bus; for PCI, see Chapter 12, PCI Interface Controller; DMA interrupts are discussed in Chapter 13, DMA Controllers; I/O causes and handling options in Chapter 15, Programmable I/O (PIO) Controller; Timer conditions and causes in Chapter 16, Timer Controller; UART conditions are defined in Chapter 17, UART Controller; SPI conditions are defined in Chapter 18, Serial Peripheral Interface.

Signal Definitions

Table 14.1 defines the signals and pins used by the interrupt controller to service and clear both RC32334 and externally generated interrupts. The internal cpu_int_n[3] signal is used by the majority of the Expansion Interrupt Pending registers and when the PCI bus writes a "1" to one of the four low order bits in the PCI_to_CPU mailbox pending register in Group 12. The pci_gnt_n[2] is mode dependent and is used as either a bus grant signal to an external device or a CPU to PCI interrupt output pin. For more information on the PCI interface controller, refer to Chapter 12, PCI Interface Controller.

Notes

Pin	Type	Function
cpu_int_n[3]	Input	CPU Interrupt #3 Negated This active-low signal is an interrupt indication to the CPU from RC32334's Interrupt Controller. **Note:** This signal is internally hooked up to the CPU's interrupt 3.
pci_gnt_n[2]	Output	PCI Bus Grant #2 Negated. Recommend external pull-up resistor. In **Host mode,** this active-low signal is an output indicating a grant to an external device. In **Satellite mode,** pci_gnt_n[2] is used as the pci_inta_n output pin. **Note**: In host mode, cpu_int_n[1] on the RC32334 can be used for a pci_inta_n input and pci_int[d:c:b]_n can use cpu_int_n[5:4:2] on the RC32334 Bus Interface. Interrupts are generated from the Expansion Interrupt Controller's Group 12.

Table 14.1 Interrupt Signal Pins and Definitions

Registers and Address Mapping

As shown in Table 14.2, each group's interrupt conditions are managed through three registers. These register functions are the same from group to group; however, the functions performed by the specific interrupt are type-specific. Group '0' (refer to Table 14.20) is a special set used as a starting point to determine which group to service. Each interrupt indicated in group '0' is also included in groups 1 through 14. The address mapping for groups 1 through 14 is provided in Tables 14.3 through 14.16.

The functional descriptions of the Interrupt Pending, Interrupt Mask, and Interrupt Clear registers are shown in Table 14.17, Table 14.18, and Table 14.19, respectively. The fields of each register are illustrated in Figures 14.3 through 14.5.

Base Address Group 0	Register Function	Offset Address	Effective Address Group 0
1800_0500	Expansion Interrupt Pending Register 0	00	Base + Offset
	Expansion Interrupt Mask Register 0	04	
	Expansion Interrupt Clear Register 0	08	

Table 14.2 Expansion Interrupt Register Group 0 Address Map

Base Address Group 1	Register Function	Offset Address	Effective Address Group 1
1800_0510	Bus Error Interrupt Pending Register 1	00	Base + Offset
	Bus Error Interrupt Mask Register 1	04	
	Bus Error Interrupt Clear Register 1	08	

Table 14.3 Bus Error Register Group 1 Address Map

Base Address Group 2	Register Function	Offset Address	Effective Address Group 2
1800_0520	PIO Low Interrupt Pending Register 2	00	Base + Offset
	PIO Low Interrupt Mask Register 2	04	
	PIO Low Interrupt Clear Register 2	08	

Table 14.4 PIO Low Register Group 2 Address Map

Notes

Base Address Group 3	Register Function	Offset Address	Effective Address Group 3
1800_0530	PIO High Interrupt Pending Register 3	00	Base + Offset
	PIO High Interrupt Mask Register 3	04	
	PIO High Interrupt Clear Register 3	08	

Table 14.5 PIO High Register Group 3 Address Map

Base Address Group 4	Register Function	Offset Address	Effective Address Group 4
1800_0540	Timer Rollover Interrupt Pending Register 4	00	Base + Offset
	Timer Rollover Interrupt Mask Register 4	04	
	Timer Rollover Interrupt Clear Register 4	08	

Table 14.6 Timer Rollover Interrupt Register Group 4 Address Map

Base Address Group 5	Register Function	Offset Address	Effective Address Group 5
1800_0550	UART 0 Interrupt Pending Register 5	00	Base + Offset
	UART 0 Interrupt Mask Register 5	04	
	UART 0 Interrupt Clear Register 5	08	

Table 14.7 UART 0 Interrupt Register Group 5 Address Map

Base Address Group 6	Register Function	Offset Address	Effective Address Group 6
1800_0560	UART 1 Interrupt Pending Register 6	00	Base + Offset
	UART 1 Interrupt Mask Register 6	04	
	UART 1 Interrupt Clear Register 6	08	

Table 14.8 UART 1 Interrupt Register Group 6 Address Map

Base Address Group 7	Register Function	Offset Address	Effective Address Group 7
1800_0570	DMA 0 Interrupt Pending Register 7	00	Base + Offset
	DMA 0 Interrupt Mask Register 7	04	
	DMA 0 Interrupt Clear Register 7	08	

Table 14.9 DMA Channel 0 Register Group 7 Address Map

Base Address Group 8	Register Function	Offset Address	Effective Address Group 8
1800_0580	DMA 1 Interrupt Pending Register 8	00	Base + Offset
	DMA 1 Interrupt Mask Register 8	04	
	DMA 1 Interrupt Clear Register 8	08	

Table 14.10 DMA Channel 1 Register Group 8 Address Map

Notes

Base Address Group 9	Register Function	Offset Address	Effective Address Group 9
1800_0590	DMA 2 Interrupt Pending Register 9	00	Base + Offset
	DMA 2 Interrupt Mask Register 9	04	
	DMA 2 Interrupt Clear Register 9	08	

Table 14.11 DMA Channel 2 Register Group 9 Address Map

Base Address Group 10	Register Function	Offset Address	Effective Address Group 10
1800_05A0	DMA 3 Interrupt Pending Register 10	00	Base + Offset
	DMA 3 Interrupt Mask Register 10	04	
	DMA 3 Interrupt Clear Register 10	08	

Table 14.12 DMA Channel 3 Register Group 10 Address Map

Base Address Group 11	Register Function	Offset Address	Effective Address Group 11
1800_05B0	PCI Controller Interrupt Pending Register 11	00	Base + Offset
	PCI Controller Interrupt Mask Register 11	04	
	PCI Controller Interrupt Clear Register 11	08	

Table 14.13 PCI Controller Interrupt Register Group 11 Address Map

Base Address Group 12	Register Function	Offset Address	Effective Address Group 12
1800_05C0	External Interrupt Pending Register 12	00	Base + Offset
	External Interrupt Mask Register 12	04	
	External Interrupt Clear Register 12	08	

Table 14.14 External Interrupt Register Group 12 Address Map

Base Address Group 13	Register Function	Offset Address	Effective Address Group 13
1800_05D0	PCI to CPU Interrupt Pending Register 13	00	Base + Offset
	PCI to CPU Interrupt Mask Register 13	04	
	PCI to CPU Interrupt Clear Register 13	08	

Table 14.15 PCI to CPU Interrupt Register Group 13 Address Map

Base Address Group 14	Register Function	Offset Address	Effective Address Group 14
1800_05E0	SPI Interrupt Pending Register 14	00	Base + Offset
	SPI Interrupt Mask Register 14	04	
	SPI Interrupt Clear Register 14	08	

Table 14.16 SPI Interrupt Register Group 14 Address Map

Notes

Interrupt Pending Register

Note that a write to any of the Pending Registers, with a Bit Field set, will set that particular Pending bit until cleared by an appropriate write to the Interrupt Clear Register. This allows software debug to test an Interrupt Service Routine (ISR), without generating the actual interrupt condition which often depends on an infrequent external condition.

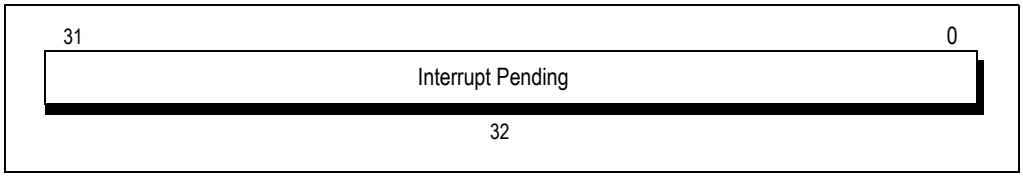


Figure 14.3 Interrupt Pending Register Fields

Bits	Field Name	Description
31:0	Pending Interrupt	Internal interrupts are registered on each rising clock edge, active low, and remain low for at least one clock cycle 1 = Interrupt pending 0 = Interrupt not pending

Table 14.17 Interrupt Pending Field Description

Interrupt Mask Register

Note that by default, RC32300 CPU core Interrupt Mask bits are set to allow interrupts but are disabled by the global enable bit being disabled by default. In contrast, RC32334 Interrupt Masks are un-set to disallow interrupts by default, in addition to having the RC32300 CPU core global enable bit disabled.

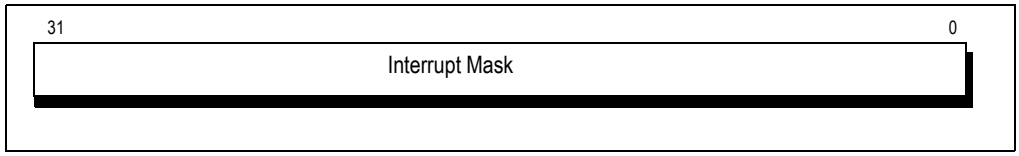


Figure 14.4 Interrupt Mask Register

Bits	Field Name	Description
31:0	Interrupt Clear	1 = Interrupt enabled/allowed 0 = Interrupt disabled/disallowed (default)

Table 14.18 Interrupt Mask Register

Interrupt Clear Register

The Interrupt Clear Register is a write only register that clears the pending interrupt bit. A masked write to the Interrupt Clear register allows a per bit clearing of all pending interrupts.

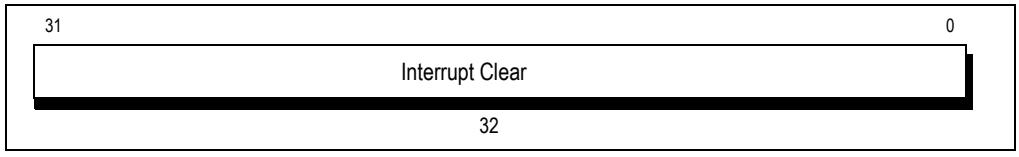


Figure 14.5 Interrupt Clear Register Field

Bits	Field Name	Description
31:0	Interrupt Mask	1 = Clear pending bit 0 = Leave pending bit unchanged (default)

Table 14.19 Interrupt Clear Register Field Descriptions

Notes

Register Group Settings

Register Group 0 Settings

Group ‘0’ is a special set of registers used as a starting point to determine which group to service. Each interrupt indicated in group ‘0’ is also included in groups 1 through 14.

Bit	Register Group 0	Expansion Register Setting
14	Interrupt Controller[14]	Indicates that the group indicated has at least one active, unmasked interrupt source
13	Interrupt Controller[13]	
12	Interrupt Controller[12]	
11	Interrupt Controller[11]	
10	Interrupt Controller[10]	
09	Interrupt Controller[9]	
08	Interrupt Controller[8]	
07	Interrupt Controller[7]	
06	Interrupt Controller[6]	
05	Interrupt Controller[5]	
04	Interrupt Controller[4]	
03	Interrupt Controller[3]	
02	Interrupt Controller[2]	
01	Interrupt Controller[1]	

Table 14.20 Group 0 Register Settings

Register Group 1 Settings

Bit	Register Group 1	Expansion Register Setting
13:01	Reserved. Must be written as ‘0’. Returns ‘0’ when read.	
00	Bus Error	**“1”** if Bus Error (**Group 1**) bits are set

Table 14.21 Group 1 (Bus Error) Register Settings

Register Group 2 Settings

Note: Only PIO pins 10:0 have a direct active low interrupt connection.

Bit	Register Group 2	Expansion Register Setting
11	PIO[10] is low	“**1**” if any of Parallel I/O (**Group 2**) bits are set
10	PIO[9] is low	
09	PIO[8] is low	
08	PIO[7] is low	
07	PIO[6] is low	
06	PIO[5] is low	
05	PIO[4] is low	
04	PIO[3] is low	
03	PIO[2] is low	
02	Reserved	
01	PIO[1] is low	
00	PIO[0] is low	

Table 14.22 Group 2 (PIO Low) Register Settings

Notes

Register Group 3 Settings

Note: Only PIO pins 6:0 have a direct active high interrupt connection.

Bit	Register Group 3	Expansion Register Setting
07	PIO[6] is high	**"1"** if any of PIO High (**Group 3**) bits are set
06	PIO[5] is high	
05	PIO[4] is high	
04	PIO[3] is high	
03	PIO[2] is high	
02	Reserved	
01	PIO[1] is high	
00	PIO[0] is high	

Table 14.23 Group 3 (PIO High) Register Settings

Register Group 4 Settings

Bit	Register Group 4	Expansion Register Setting
07	Timer7 Rollover Interrupt for ColdReset	**"1"** if any of the Timer Rollover Interrupt (**Group 4)** bits are set
06	Timer6 Rollover Interrupt for DramRefresh	
05	Timer5 Rollover Interrupt for IP BusTimeout (BusError)	
04	Timer4 Rollover Interrupt for CPU BusTimeout (BusError)	
03	Timer3 Rollover Interrupt for Watchdog (uses ColdReset_n instead of Reset_n)	
02	Timer2 Rollover Interrupt	
01	Timer1 Rollover Interrupt	
00	Timer0 Rollover Interrupt	

Table 14.24 Group 4 (Timer Rollover Interrupt) Register Settings

Register Group 5 Settings

Bit	Register Group 5	Expansion Register Setting
02	UART0 Interrupt 2 RxRdy	**"1"** if any of the UART0 (**Group 5**) bits are set
01	UART0 Interrupt 1 TxRdy	
00	UART0 Interrupt 0 IIR(0)	

Table 14.25 Group 5 (UART 0 Interrupt) Register Settings

Register Group 6 Settings

Bit	Register Group 6	Expansion Register Setting
02	UART1 Interrupt 2 RxRdy	**"1"** if any of the UART1 (**Group 6)** bits are set
01	UART1 Interrupt 1 TxRdy	
00	UART1 Interrupt 0 IIR(0)	

Table 14.26 Group 6 (UART 1 Interrupt) Register Settings

Notes

Register Group 7 Settings

Bit	Register Group 7	Expansion Register Setting
04	DMA Ch0 DMA Clear Interrupt	**"1"** if any of the DMA Ch0 Interrupt 0 (**Group 7)** bits are set
03	DMA Ch0 DMA Transaction Complete	
02	DMA Ch0 Descriptor Not Owned Error Interrupt	
01	DMA Ch0 End Too Early Error Interrupt	
00	DMA Ch0 Done Interrupt	

Table 14.27 Group 7 (DMA Memory2I/O Interrupt 0) Register Settings

Register Group 8 Settings

Bit	Register Group 8	Expansion Register Setting
04	DMA Ch1 DMA Clear Interrupt	**"1"** if any of the DMA Ch1 Interrupt 1 (**Group 8)** bits are set
03	DMA Ch1 DMA Transaction Complete	
02	DMA Ch1 Descriptor Not Owned Error Interrupt	
01	DMA Ch1 End Too Early Error Interrupt	
00	DMA Ch1 Done Interrupt	

Table 14.28 Group 8 (DMA Memory2IO Interrupt 1) Register Settings

Register Group 9 Settings

Bit	Register Group 9	Expansion Register Setting
04	DMA Ch2 DMA Clear Interrupt	**"1"** if any of the DMA Ch2 Interrupt 0 (**Group 9)** bits are set
03	DMA Ch2 DMA Transaction Complete	
02	DMA Ch2 Descriptor Not Owned Error Interrupt	
01	DMA Ch2 End Too Early Error Interrupt	
00	DMA Ch2 Done Interrupt	

Table 14.29 Group 9 (DMA PCI Master Interrupt 0) Register Settings

Register Group 10 Settings

Bit	Register Group 10	Expansion Register Setting
04	DMA Ch3 DMA Clear Interrupt	**"1"** if any of the DMA Ch3 Interrupt 1 (**Group 10)** bits are set
03	DMA Ch3 DMA Transaction Complete	
02	DMA Ch3 Descriptor Not Owned Error Interrupt	
01	DMA Ch3 End Too Early Error Interrupt	
00	DMA Ch3 Done Interrupt	

Table 14.30 Group 10 (DMA PCI Master Interrupt 1) Register Settings

Register Group 11 Settings

Group 11 pending interrupts indicate a PCI controller error condition as detailed in Table 12.10 in the PCI Controller Interrupt Pending Register 11 section of Chapter 12, PCI Interface Controller.

Notes

Bit	Register Group 11	Expansion Register Setting
03	PCI Controller Interrupt 3	**"1"** if any of the PCI Controller (**Group 11)** bits are set
02	PCI Controller Interrupt 2	
01	PCI Controller Interrupt 1	
00	PCI Controller Interrupt 0	

Table 14.31 Group 11 (PCI Controller) Register Settings

Register Group 12 Settings

When PCI is in the Satellite Mode, Pending Interrupts in Register 12 affect the PCI Interrupt pin. This register does not affect the internal cpu_int_n[3] signal directly. The output always goes to bit 12 of the Interrupt0 Register, which then may be masked/unmasked to cause the internal cpu_int_n[3] signal to assert.

Note: The pci_interrupt_n (pci_gnt_n[2]) signal is internally synchronized with the pci_clk signal twice. And as such the output propagation is relative to the rising edge of pci_clk instead of cpu_masterclk.

Bit	Register Group 12	Expansion Register Setting
15	PCI CPU2PCI Mailbox Interrupt 3	**"1"** if any of the External Interrupt (**Group 12)** bits are set
14	PCI CPU2PCI Mailbox Interrupt 2	
13	PCI CPU2PCI Mailbox Interrupt 1	
12	PCI CPU2PCI Mailbox Interrupt 0	
11	DMA MEM2IO Descriptor Not Owned Error Interrupt 1	
10	DMA MEM2IO End Too Early Error Interrupt 1	
09	DMA MEM2IO Done Interrupt 1	
08	DMA MEM2IO Descriptor Not Owned Error Interrupt 0	
07	DMA MEM2IO End Too Early Error Interrupt 0	
06	DMA MEM2IO Done Interrupt 0	
05	UART1 Interrupt 2[1]	
04	UART1 Interrupt 1[1]	
03	UART1 Interrupt 0[1]	
02	UART0 Interrupt 2	**"1"** if any of the External Interrupt (**Group 12)** bits are set
01	UART0 Interrupt 1	
00	UART0 Interrupt 0	

Table 14.32 Group 12 Register Settings

1. Not in RC32332.

Register Group 13 Settings

Group 13 pending interrupts indicate a PCI controller PCI initiated interrupt to the RC32334 CPU, as detailed in Chapter 12, Table 12.12.

Notes

Bit	Register Group 13	Expansion Register Setting
03	PCI PCI2CPU Mailbox Interrupt 3	**"1"** if any of the PCI PCI12CPU Mailbox (**Group 13)** bits are set
02	PCI PCI2CPU Mailbox Interrupt 2	
01	PCI PCI2CPU Mailbox Interrupt 1	
00	PCI PCI2CPU Mailbox Interrupt 0	

Table 14.33 Group 13 Register Settings

Register Group 14 Settings

Bit	Register Group 14	Expansion Register Setting
00	SPI Interrupt 0	**"1"** if the SPI (**Group 14)** bits are set

Table 14.34 Group 14 Register Settings

Timing Diagrams

For the timing of various transactions asserting/de-asserting internal cpu_int_n[3], see Figures 14.6 through 14.9. The timing behaviors of transactions asserting/de-asserting PCI are shown in Figures 14.10 and 14.11. The timing requirements for cpu_int_n[5,4,2,1,0] and cpu_nmi_n are shown in Figure 14.12.

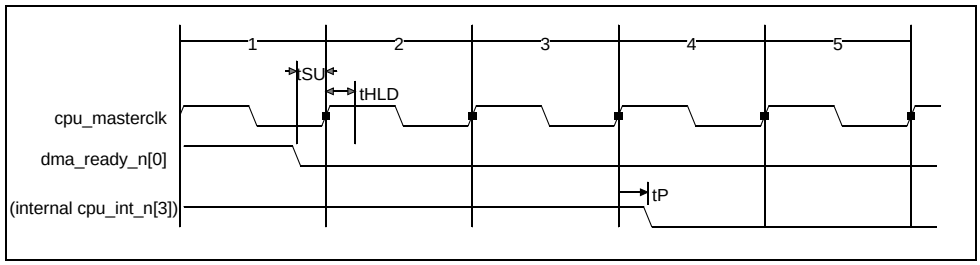


Figure 14.6 PIO Input Asserting Internal cpu_int_n[3]

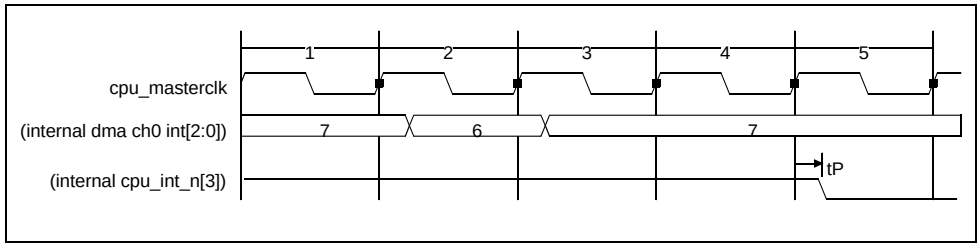


Figure 14.7 Internal Condition Asserting Internal cpu_int_n[3] Interrupt

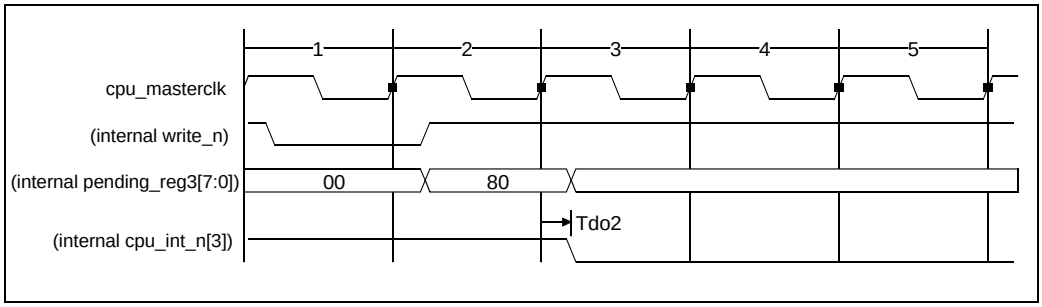


Figure 14.8 Pending Register Write Asserting Internal cpu_int_n[3]

Notes

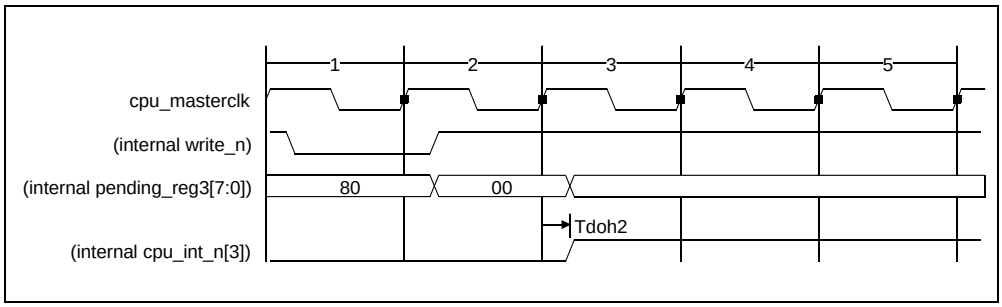


Figure 14.9 Pending or Clear Register Write De-Asserting Internal cpu_int_n[3] Interrupt

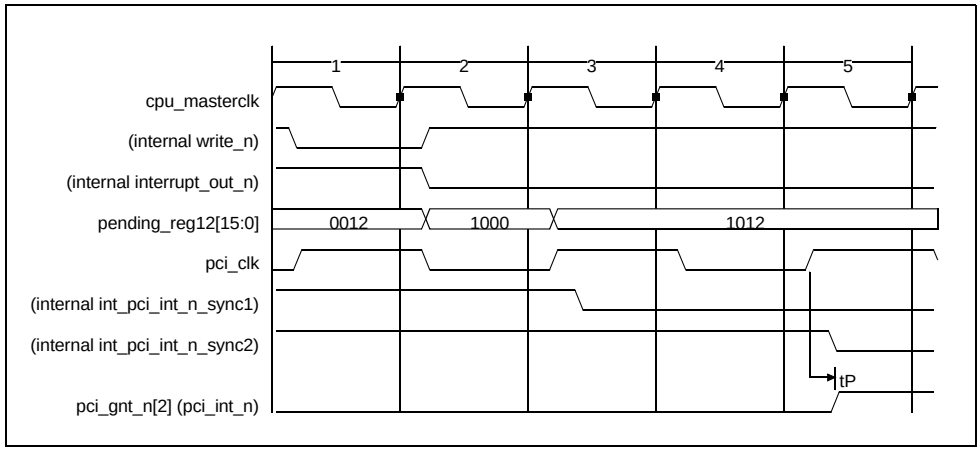


Figure 14.10 Internal Condition Asserting PCI Interrupt

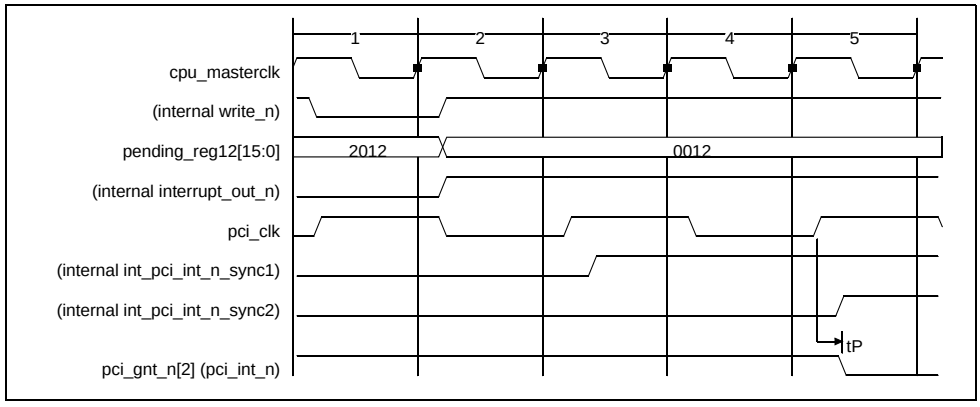


Figure 14.11 Pending or Clear Register Write De-Asserting PCI Interrupt

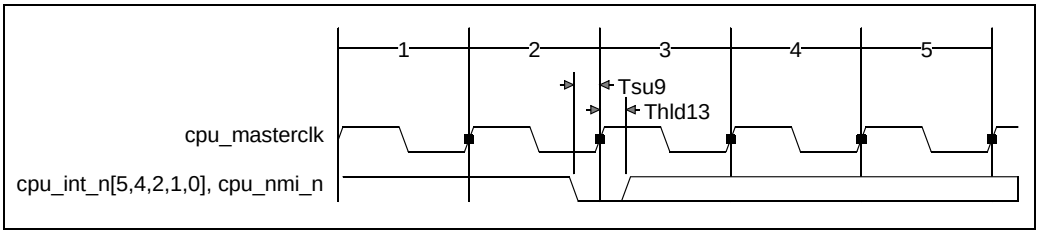


Figure 14.12 CPU Interrupts

Notes

RC32334 Interrupt Flow

1. Initialize Interrupts

1. Disable CPU CP0 Status Register Global Interrupt Enable Bit.
2. Enable CPU CP0 Status Register Interrupt Mask Bit 3. (Optionally disable the other seven CPU interrupts.)
3. Enable the appropriate RC32334 Expansion Interrupt Mask Register Bit. (Optionally disable the other interrupt mask bits.)
4. Clear the appropriate RC32334 Expansion Interrupt Clear Register Bit (for all unmasked interrupt bits).
5. Enable CPU CP0 Status Register Global Interrupt Enable Bit.

2. Wait for Interrupt

1. Hardware Interrupt generated by pulsing the appropriate signal/pin low for at least 1.0 clock, either internally or externally; or by Software writing to the appropriate Pending Interrupt Register Bit.
2. The RC32334 Expansion Interrupt hardware will set the appropriate Expansion Interrupt Pending Bit. The Pending Bit will remain set until Software clears it.
3. If the appropriate Expansion Interrupt Mask Bit is not set, then no further hardware action occurs, otherwise an internal interrupt is sent to Expansion Interrupt Register set 0 and to the RC32334 cpu_int_n pin.
4. The internal cpu_int_n signal asserts and on a clock by clock basis asserts the internal CPU interrupt port causing it to assert the CPU CP0 Cause Register Interrupt Pending Bit 3. The cpu_int_n pin remains asserted until Software clears the appropriate Expansion Interrupt Pending Bit (or disables the appropriate Expansion Interrupt Mask bit).
5. If the CPU CP0 Status Register Interrupt Mask Bit 3 is enabled, then CPU takes exception and jumps to Exception Vector. CPU CP0 Status Register Global Interrupt Enable Bit is automatically disabled.

3. Software Interrupt Service Routine (ISR)

1. If Software ISR read of CPU CP0 Cause Register Interrupt Pending Bit 3 is set, then continue with ISR.
2. If Software ISR read of the appropriate RC32334 Expansion Interrupt Pending Register Bit is
3. set, then continue with ISR.
4. Clear the appropriate interrupt source (device dependent, for instance read UART data), causing the interrupt source to become de-asserted.
5. Clear the appropriate RC32334 Expansion Interrupt Clear Register Bit. If no other non-masked interrupts exist, this will cause the RC32334 pin to de-assert.
6. Either check for more interrupts or return from exception (ERET instruction automatically re-enables CPU CP0 Status Register Global Interrupt Enable Bit).

Optional Algorithm for Priority Interrupts

The first Expansion Interrupt Register set 0 combines the interrupt output from each of the other Expansion Interrupt Register sets. If Expansion Register set 0 is used, then the Software ISR can more quickly find the cause of an expansion interrupt by then jumping directly to the Expansion Interrupt Pending Register that has a Register set 0 Interrupt Pending bit pending. In this case, after receiving the interrupt, do the following:

1. Service all the interrupts associated with the Expansion Interrupt Set and clear the original causes.
2. Clear the Interrupt Pending bits using the Expansion Interrupt set's Interrupt Clear Register.
3. Clear the appropriate Expansion Interrupt Register set 0 entry using Interrupt Clear Register 0.

Optional Algorithm for Non-Prioritized Interrupts

The first Expansion Interrupt Register 0 set can be ignored (masked out) in which case the Software ISR simply checks each Expansion Interrupt Pending Register in a linear search.

Notes

Programmable I/O (PIO) Controller

Notes

Introduction

The RC32334 provides software programmable I/O (PIO) pins, so that unused pins can be used as general purpose discrete I/O pins. As such, if a pin's function—for example, Timer Output—is not required, then that pin can be programmed for use as a general purpose PIO pin.

Once programmed to a general purpose function, pins can then be software programmed as inputs or outputs. When set in the output mode, the pin's value becomes software programmable. When set to the input mode, the pins are software readable. This chapter provides the signal descriptions, register mapping, and programming information needed to use this software programmable feature.

Features

- *16[1] peripheral pins, reusable as PIO pins*
- *Bidirectional pins*
  - *Output pins can be programmed hi/low, in parallel*
  - *Input pins can be read in parallel.*

Overview

The RC32334's PIO pins are programmable in both the input and output directions. When programming to the input mode, data from the input pin are read by the RC32300 CPU core as required. In the output mode, data can also control the output level of the pin at any time. The default state of most pins is input.

PIO pins are multiplexed between peripheral and general purpose use, as shown in the signal definition tables that follow. As such, PIO pins on unused peripherals can be reused on a system basis for the following general purpose uses:

- *as a parallel port*
- *as an interrupt input/output from or to another device*
- *as status input/output from or to another device.*

Switching between the four possible modes is accomplished through the following general algorithm:

1. Optional reset initialization by use of external pull-ups/pull-downs.
2. Write the PIO Direction Register bits to be in the input mode.
3. Write the PIO Function Register bits to the desired mode.
4. Write the PIO Direction Register bits to the desired mode.
5. The PIO Data register is ready for reading and writing and the internal peripherals are ready.

1. The RC32332 includes 12 PIO pins.

Notes

Block Diagram

Figure 15.1 shows the PIO block diagram and Figure 15.2 shows the PIO bit-slice block diagram.

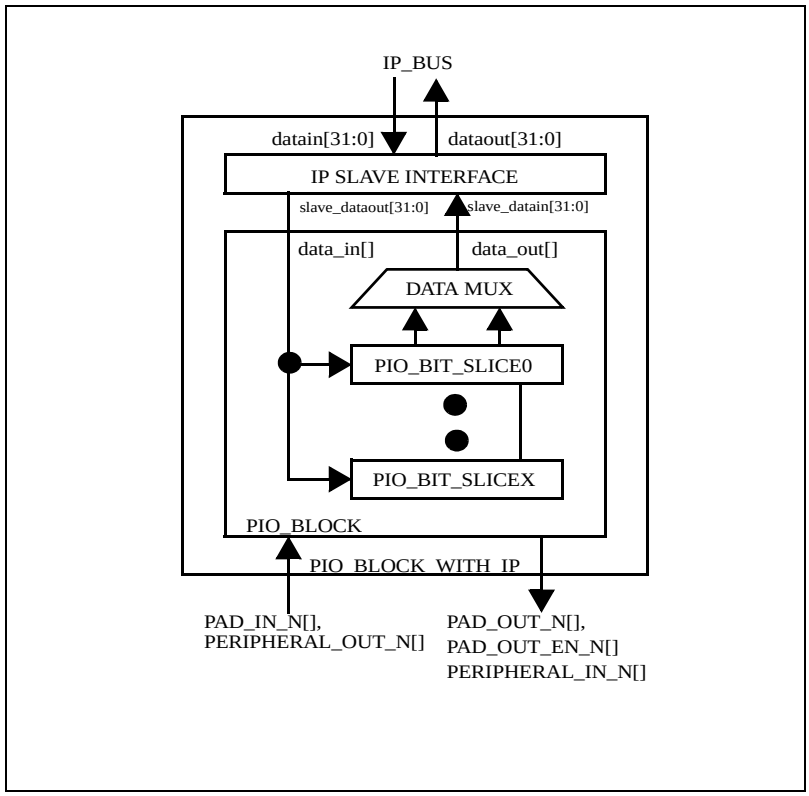


Figure 15.1 PIO Block Diagram

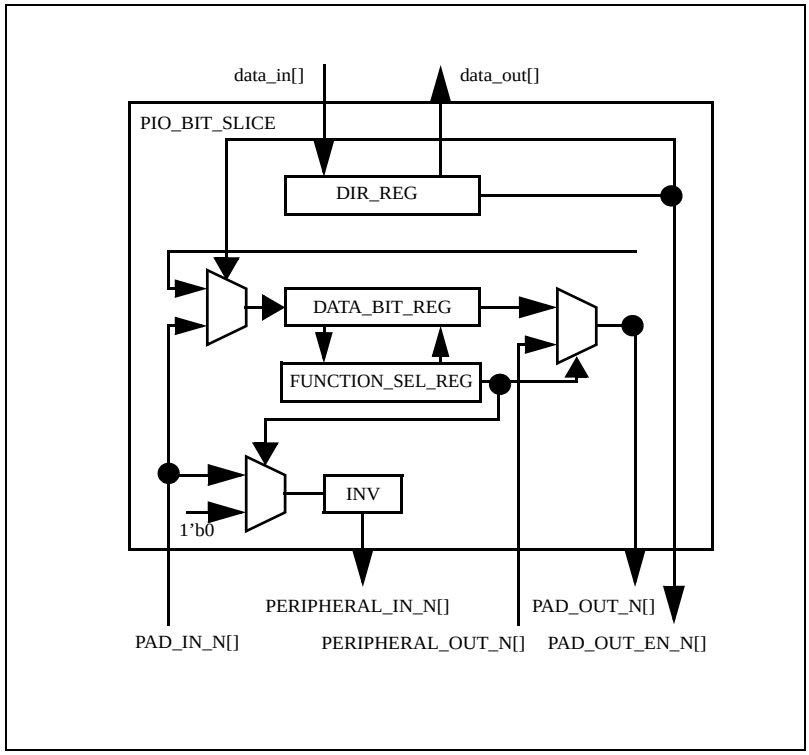


Figure 15.2 PIO Block Diagram Bit-Slice

Notes

Performing Initialization Programming

Peripheral Function Input mode: At reset, the Input mode is the default state for most of the pins; therefore, no additional programming is necessary and the PIO pins are ready for use by the internal peripheral. However, in the case where the default is set to the Output mode, perform the following steps:

1. Write the PIO Direction register bits to be in the Input mode.
2. The PIO Function bits are already in the Function mode.
3. The PIO pins are ready for use by the internal peripheral.

Peripheral Function Output mode: At reset, some pins may be defaulted to the output mode and are ready for use by the internal peripheral. However, in the case where the default has been set to the Input mode, perform the following steps:

1. Optional reset initialization by use of external pull-ups/pull-downs.
2. Write the PIO Direction Register bits to be in the Output mode.
3. The PIO Function Register bits are already in the Function mode.
4. The PIO pins are ready for use by the internal peripheral.

General Purpose Input mode: Program the unused pins for use in the general purpose Input mode function using the following steps:

1. Write the PIO Function Register bits to the General Purpose mode, so that unused internal peripheral ports will be internally driven to their de-asserted value.
2. If the pin is an output by default, write the PIO Direction Register bits to be in the Input mode.
3. The PIO Data Register is ready for reading.

General Purpose Output mode: Program unused pins for use in the general purpose output mode function using the following steps:

1. Initialize optional reset by use of external pull-ups/pull-downs.
2. Write the PIO Function Register bits to be in the General Purpose mode.
3. Write the PIO Direction Register bits to be in the Output mode.
4. The PIO Data Register is ready for writing.

Note 1: Pins that are not in the general purpose output mode automatically mask their respective Data Register bits from being written.

Note 2: When switching from the input mode to the output mode, the output will initially drive the value registered by the Data Register, 1 clock previous to the input to output transition.

Signal Definitions

The signals listed in the tables that follow control the Serial Mode Protocol, UART Interface, Timer and DMA Interface functions. Any active-low signals are noted by an _n. The alternate pin names—including PIO multiplexed pins—and descriptions are also listed next to the main signal name. For a summary of the differences between alternate PIO names in the RC32334 and RC32332, refer to Appendix G, Tables G.2 and G.3.

SPI Interface	Type	Alternate	Descriptions
spi_mosi	I/O	pio[10]	**SPI Data Output** Serial mode: Output pin from RC32334 as an Input to a Serial Chip for the Serial data input stream. PCI satellite mode: Output pin from RC32334 that connects as an Input to a Serial Chip for the Serial data input stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Alternate function: PIO[10]. Defaults to the output direction at reset time.

Table 15.1 Serial Mode Protocol/Alternate Signal Descriptions (Part 1 of 2)

Notes

SPI Interface	Type	Alternate	Descriptions
spi_miso	I/O	pio[7]	**SPI Data Input** Serial mode: Input pin to RC32334 from the Output of a Serial Chip for the Serial data output stream. PCI satellite mode: Input pin from RC32334 that connects as an output to a Serial Chip for the Serial data output stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Defaults to input direction at reset time. Alternate function: PIO[7].
spi_sck	I/O	pio[9]	**SPI Clock** Serial mode: Output pin for Serial Clock. PCI satellite mode: Output pin for Serial Clock for loading PCI Configuration Registers in the RC323334 Reset Initialization Vector PCI boot mode. Alternate function: PIO[9]. Defaults to the output direction at reset time.
spi_ss_n	I/O	pio[8]	**SPI Chip Select** Output pin selecting the serial protocol device as opposed to the PCI satellite mode EEPROM device. Alternate function: PIO[8]. Defaults to the output direction at reset time.

Table 15.1 Serial Mode Protocol/Alternate Signal Descriptions (Part 2 of 2)

16550 UART Interface	Type	Alternate Signals	Descriptions
uart_rx[0] uart_rx[1]	I/O	pio[6] pio[4]	UART Receive Data Bus UART mode: Each UART channel receives data on their respective input pin.
uart_tx[0] uart_tx[1]	I/O	pio[5] pio[3]	UART Transmit Data Bus Recommend external pull-up. UART mode: Each UART channel sends data on their respective output pin. Note that these pins default to inputs at reset time and must be programmed via the PIO interface before being used as UART outputs.
uart_cts_n[0][1] uart_dsr_n[0][1] uart_dtr_n[0][1] uart_rts_n[0][1]	I/O I/O I/O I/O	pio[15] pio[14] pio[13] pio[12]	UART Transmit Data Bus UART mode: Data bus modem control signal pins for UART channel 0 PIO mode: These pins are also multiplexed as PIO pins.

Table 15.2 UART Interface/Alternate Signal Descriptions

1. Not in the RC32332.

Timer/Counter	Type	Alternate Signal	Description
timer_tc_n[0][1]	I/O	pio[2]	Timer Terminal Count Overflow Negated Output indicating that the timer has reached its count compare value and has overflowed back to zero.

Table 15.3 Timer/Alternate Signal Descriptions

1. Not in the RC32332.

Notes

DMA Interface	Type	Alternate Signal	Description
dma_ready_n[0] dma_ready_n[1][1]	I/O	pio[1], dma_done_n[0] pio[0], dma_done_n[1][1]	DMA Ready Negated Bus Requires external pull-up. Input pin for general purpose DMA channels[1:0] that can initiate the next data in the current DMA descriptor frame.
dma_done_n[1:0]	I/O	dma_ready_n[1:0]	DMA Done Requires external pull-up. Input pin for general purpose DMA channels[1:0] that can terminate the current DMA descriptor frame.

Table 15.4 DMA Interface/Alternate Signal Descriptions

1. Not in the RC32332.

PCI Interface	Type	Alternate Signal	Description
pci_eeprom_cs	I/O	pci_gnt_n[1], pio[11]	PCI EEPROM Chip Select
pci_gnt_n[1]	I/O	pci_eeprom_cs, pio[11]	PCI Bus Grant # 1 Negated

Table 15.5 PIO Interface/Alternate Signal Descriptions

Register Mapping and Definitions

Programming PIO pins to be used in the Peripheral Function Input/Output or General Purpose Input/Output modes is handled through initialization and setup of the PIO Data, PIO Direction Control, and PIO Function Control registers. The address mapping for each register is listed in Table 15.6. Each register's description includes the default value.

Base Address	Register Name	Offset Address	Effective Address
1800_0600	PIO Data Register 0	00	Base + Offset
	PIO Direction Control Register 0	04	
	PIO Effect Select Control Register 0	08	
1800_0610	PIO Data Register 1	00	
	PIO Direction Control Register 1	04	
	PIO Effect Select Control Register 1	08	
1800_060C	PIO New Feature Register 0	0C	
1800_060C	PIO New Feature Register 1	1C	

Table 15.6 PIO Register Address Map

PIO register set 0 adds alternative PIO usage to the SPI, UART data, timer, and DMA pins.

PIO register set 1 adds alternative PIO usage to the modem control of UART0. Added are: uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], uart_rts_n[0] pins via PIO register set 1 (starting at physical address 0x18000610) bits 4:1 respectively. All modem signals default to inputs. Systems may optionally use external pull-ups or pull-downs to initialize pins that are to be used as outputs.

PIO register set 1 also adds PCI EEPROM writeability to the PCI EEPROM by allowing pci_eeprom_cs to be controlled from a software bit-blasting driver. The pci_eeprom_cs signal defaults to an output.

PIO Data Register 0

Bits in this register clock data from the pins, if set in the input direction or the special function mode. Bits can only be written if that bit is in both the output direction and general purpose mode. Figure 15.3 illustrates the fields of PIO Data Register 0, and Tables 15.7 and 15.8 describe the fields.

Notes

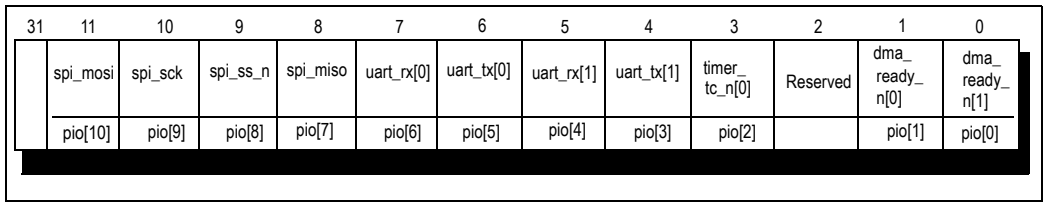


Figure 15.3 PIO Data Register 0 Fields

Note: timer_tc_n[0], uart_rx[1], and uart_tx[1], shown in Figure 15.3, are not in the RC32332.

Bit	Field	Description
31:12	Reserved to 1	Requires 1 to be written in these fields
11	spi_mosi pio[10]	SPI control functions brought to external pins
10	spi_sck pio[9]	
9	spi_ss_n pio[8]	
8	spi_miso pio[7]	
7	uart_rx[0] pio[6]	UART data brought to external pins
6	uart_tx[0] pio[5]	
5	uart_rx[1][1] pio[4]	
4	uart_tx[1][1] pio[3]	
3	timer_tc_n[0][1] pio[2]	Timer function brought to external pin
2	Reserved to 1	Requires 1 to be written to this field
1	dma_ready_n[0] pio[1]	DMA control functions brought to external pins
0	dma_ready_n[1] pio[0]	

Table 15.7 PIO Data Register 0 Field Description

1. Not in the RC32332.

Bit	Description	
31:0	PIO Data Register 0	
	Value	**Description**
	1	PIO pin is high (default)
	0	PIO pin is low

Table 15.8 PIO Data Register 0 High/Low Descriptions

PIO Data Register 1

Bits in this register clock data from the pins, if set in the input direction or the special function mode. Bits can only be written if that bit is in both the output direction and general purpose mode. Figure 15.4 illustrates the fields of PIO Data Register 1, and Tables 15.9 and 15.10 describe the fields.

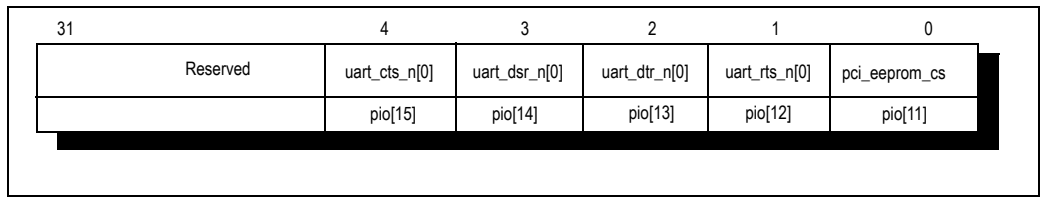


Figure 15.4 PIO Data Register 1 Fields

Notes

Note: uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], and uart_rts_n[0], shown in Figure 15.4, are not in the RC32332.

Bit	Field	Description
31:5	Reserved to 1	Requires 1 to be written in these fields
4	uart_cts_n[0][1] pio[15]	UART modem control functions brought to external pins
3	uart_dsr_n[0][1] pio[14]	
2	uart_dtr_n[0][1] pio[13]	
1	uart_rts_n[0][1] pio[12]	
0	pci_eeprom_cs pio[11]	PCI EEPROM chip select brought to external pin

Table 15.9 PIO Data Register 1 Field Description

1. Not in the RC32332.

Bit	Description	
31:0	PIO Data Register 1	
	Value	**Description**
	1	PIO pin is high (default)
	0	PIO pin is low

Table 15.10 PIO Data Register 1 High/Low Descriptions

PIO Direction Register 0

This 32-bit register programs the Input/Output modes for both the general purpose and special function modes. When programmed to the input mode, data from the input pin can be read by the RC32300 CPU core as required. When in the output mode, data can also control the output level of the pin at any time. Figure 15.5 illustrates the fields of PIO Direction Register 0, and Tables 15.11 and 15.12 describe the fields.

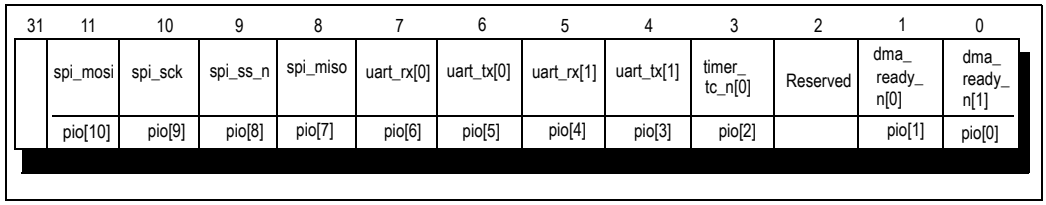


Figure 15.5 PIO Direction Register 0 Fields

Note: timer_tc-n[0], uart_rx[1], and uart_tx[1], shown in Figure 15.5, are not in the RC32332.

Bit	Field	Description
31:12	Reserved to 0	Requires 0 to be written in these fields
11	spi_mosi pio[10]	SPI control functions brought to external pins
10	spi_sck pio[9]	
9	spi_ss_n pio[8]	
8	spi_miso pio[7]	

Table 15.11 PIO Function Direction Register 0 Field Description (Part 1 of 2)

Notes

Bit	Field	Description
7	uart_rx[0] pio[6]	UART data brought to external pins
6	uart_tx[0] pio[5]	
5	uart_rx[1][1] pio[4]	
4	uart_tx[1][1] pio[3]	
3	timer_tc_n[0][1] pio[2]	Timer function brought to external pin
2	Reserved to 0	Requires 0 to be written in this field
1	dma_ready_n[0] pio[1]	DMA control functions brought to external pins
0	dma_ready_n[1] pio[0]	

Table 15.11 PIO Function Direction Register 0 Field Description (Part 2 of 2)

1. Not in the RC32332.

Bit	Description
31:0	PIO Direction Register 0

Value	Description
1	PIO pin is an output (default for pci_eeprom_cs, spi_mosi, spi_sck, spi_cs)
0	PIO pin is an input (default for most pins)

Table 15.12 PIO Direction Register 0 Input/Output Descriptions

PIO Direction Register 1

This 32-bit register programs the Input/Output modes for both the general purpose and special function modes. When programmed to the input mode, data from the input pin can be read by the RC32300 CPU core as required. When in the output mode, data can also control the output level of the pin at any time. Figure 15.6 illustrates the fields of PIO Direction Register 1, and Tables 15.13 and 15.14 describe the fields.

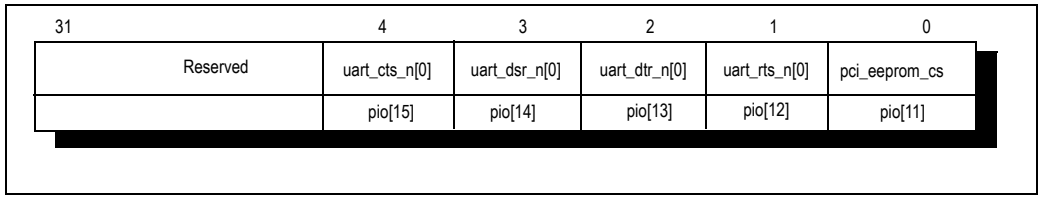


Figure 15.6 PIO Direction Register 1 Fields

Note: uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], and uart_rts_n[0], shown in Figure 15.6, are not in the RC32332.

Notes

Bit	Field	Description
31:5	Reserved to 0	Requires 0 to be written in these fields
4	uart_cts_n[0][1] pio[15]	UART modem control functions brought to external pins
3	uart_dsr_n[0][1] pio[14]	
2	uart_dtr_n[0][1] pio[13]	
1	uart_rts_n[0][1] pio[12]	
0	pci_eeprom_cs pio[11]	PCI EEPROM chip select brought to external pin

Table 15.13 PIO Direction Register 1 Field Description

1. Not in the RC32332.

Bit	Description
31:0	PIO Direction Register 1

Value	Description
1	PIO pin is an output (default for pci_eeprom_cs, spi_mosi, spi_sck, spi_cs)
0	PIO pin is an input (default for most pins)

Table 15.14 PIO Direction Register 1 Input/Output Description

PIO Function Select Register 0

When in the input direction, the pin goes to the general purpose data bit regardless of the value in the Function Select Field; however, if the Function Select Field is selected, the pin also goes to the internal module. If the Function Select Field is not selected, then the internal module input is held de-asserted high. When in the output direction, the pin is generated from either the internal module or the data register, depending upon the value of the PIO Function Select Bit Field. Figure 15.7 illustrates the fields of PIO Function Select Register 0, and Tables 15.15 and 15.16 describe the fields.

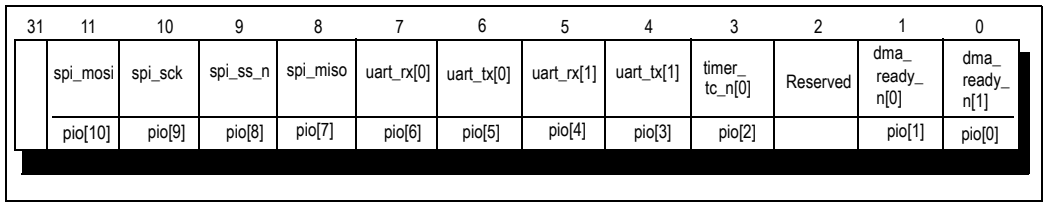


Figure 15.7 PIO Function Select Register 0 Fields

Note: timer_tc_n[0], uart_rx[1], and uart_tx[1], shown in Figure 15.7, are not in the RC32332.

Bit	Field	Description
31:12	Reserved to 1	Requires 1 to be written in these fields
11	spi_mosi pio[10]	SPI control functions brought to external pins
10	spi_sck pio[9]	
9	spi_ss_n pio[8]	
8	spi_miso pio[7]	

Table 15.15 PIO Function Select Register 0 Field Description (Part 1 of 2)

Notes

Bit	Field		Description
7	uart_rx[0]	pio[6]	UART data brought to external pins
6	uart_tx[0]	pio[5]	
5	uart_rx[1][1]	pio[4]	
4	uart_tx[1][1]	pio[3]	
3	timer_tc_n[0][1]	pio[2]	Timer function brought to external pin
2	Reserved to 1		Requires 1 to be written in this field
1	dma_ready_n[0]	pio[1]	DMA control functions brought to external pins
0	dma_ready_n[1]	pio[0]	

Table 15.15 PIO Function Select Register 0 Field Description (Part 2 of 2)

1. Not in the RC32332.

Bit	Description
31:0	PIO Function Select Register 0

Value	Description
1	PIO pin is a special function pin connected to/from an internal module (default)
0	PIO pin is a general purpose pin

Table 15.16 PIO Special Function/General Purpose Select Register 0 Description

PIO Function Select Register 1

When in the input direction, the pin goes to the general purpose data bit regardless of the value in the Function Select Field; however, if the Function Select Field is selected, the pin also goes to the internal module. If the Function Select Field is not selected, then the internal module input is held de-asserted high. When in the output direction, the pin is generated from either the internal module or the data register, depending upon the value of the PIO Function Select Bit Field. Figure 15.7 illustrates the fields of the PIO Function Select Register 1, and Tables 15.17 and 15.18 describe the fields.

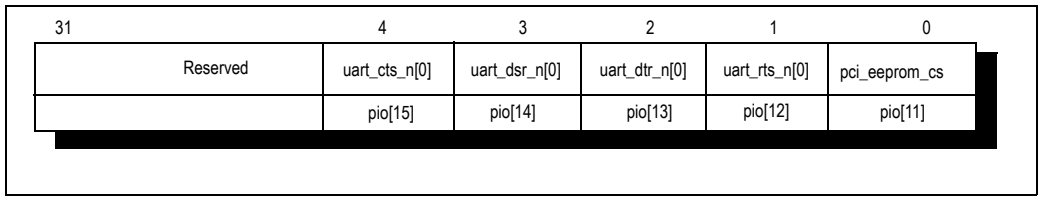


Figure 15.8 PIO Function Select Register 1 Fields

Note: uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], and uart_rts_n[0], shown in Figure 15.8, are not in the RC32332.

Notes

Bit	Field	Description
31:5	Reserved to 1	Requires 1 to be written in these fields
4	uart_cts_n[0][1] pio[15]	UART modem control functions brought to external pins
3	uart_dsr_n[0][1] pio[14]	
2	uart_dtr_n[0][1] pio[13]	
1	uart_rts_n[0][1] pio[12]	
0	pci_eeprom_cs pio[11]	PCI EEPROM chip select brought to external pin

Table 15.17 PIO Function Select Register 1 Field Description

1. Not in the RC32332.

Bit	Description
31:0	PIO Function Select Register 1

Value	Description
1	PIO pin is a special function pin connected to/from an internal module (default)
0	PIO pin is a general purpose pin

Table 15.18 PIO Function Select Register 1 Special Function/General Purpose Description

New Feature Register

New Feature Register 0

When the New Feature field is selected, the entire group of pins associated with PIO Register Set 0 becomes synchronized with double register sampling using the system clock.

New Feature Register 1

When the New Feature field is selected, the entire group of pins associated with PIO Register Set 1 becomes synchronized with double register sampling using the system clock.

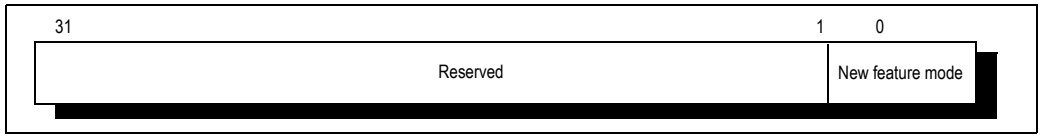


Figure 15.9 PIO New Feature Register Fields

Bits	Field Name	Description
31:1	Reserved to 0	Requires 0 to be written in these fields
0	New Feature Mode	

Value	Description
1	New Feature mode: Adds double registering synchronization on external inputs.
0	Backward compatibility mode

Table 15.19 PIO New Feature Register Field Description

Notes

Timing Diagrams

In Figure 15.10 and Figure 15.11, timing for pio[7:6] is shown. Note that the timing for all other PIO signals, pio[15:8] and pio[5:0], is similar when the appropriate PIO Data Register and its bit fields are read or written.

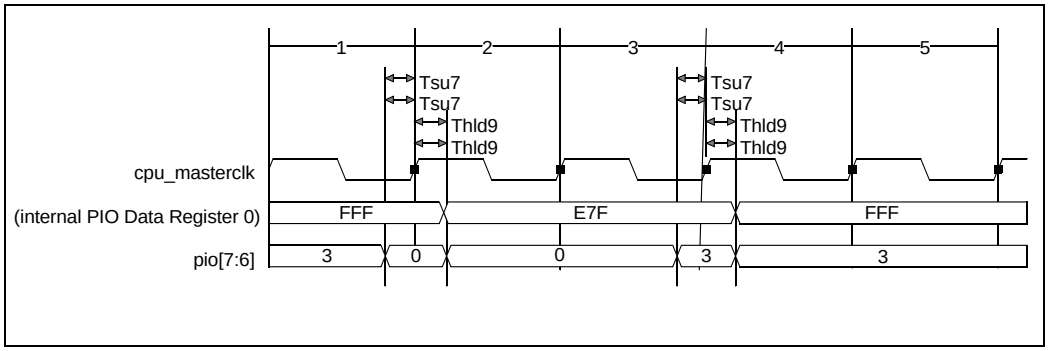


Figure 15.10 PIO Input, Affecting Data Register

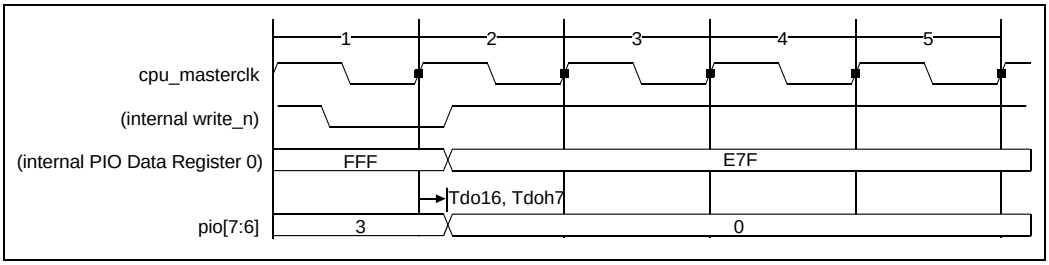


Figure 15.11 Data Register Write, Affecting PIO Output

Timer Controller

Notes

Introduction

In addition to the timer on the RC32300 CPU core, the RC32334 has eight on-chip timers: Three general purpose timers and five timers that are optionally dedicated to Watchdog, CPU bus timeout, IP bus timeout, SDRAM refresh, and WarmReset. These eight timers are different and in addition to the timer available on the RC32300 CPU core as part of CP0. These eight system timers count on each system clock beginning from zero, timing out after reaching a programmable compare value and resetting to zero automatically. Uses for these timers include real-time clock, cascaded real-time clock and time-slice clock.

Features

- *3 general purpose 32-bit timers*
- *5 8/16-bit peripheral dedicated timers available for general reuse*
- *Programmable compare/count roll over value*
- *Selectable count mode versus input gate mode for timer0 and timer1*
- *Timer 0 internally cascaded to Timer 1.*

Block Diagram

Figure 16.1 and Figure 16.2 show the Timing block diagram and Individual Timer Core block diagram, respectively.

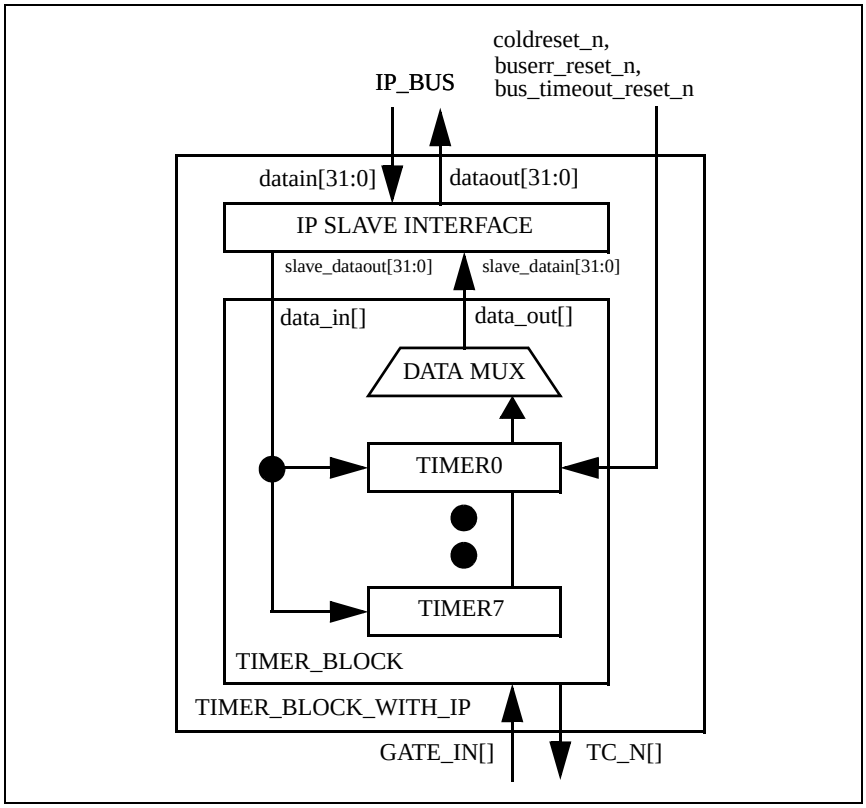


Figure 16.1 Timer Block Diagram

Notes

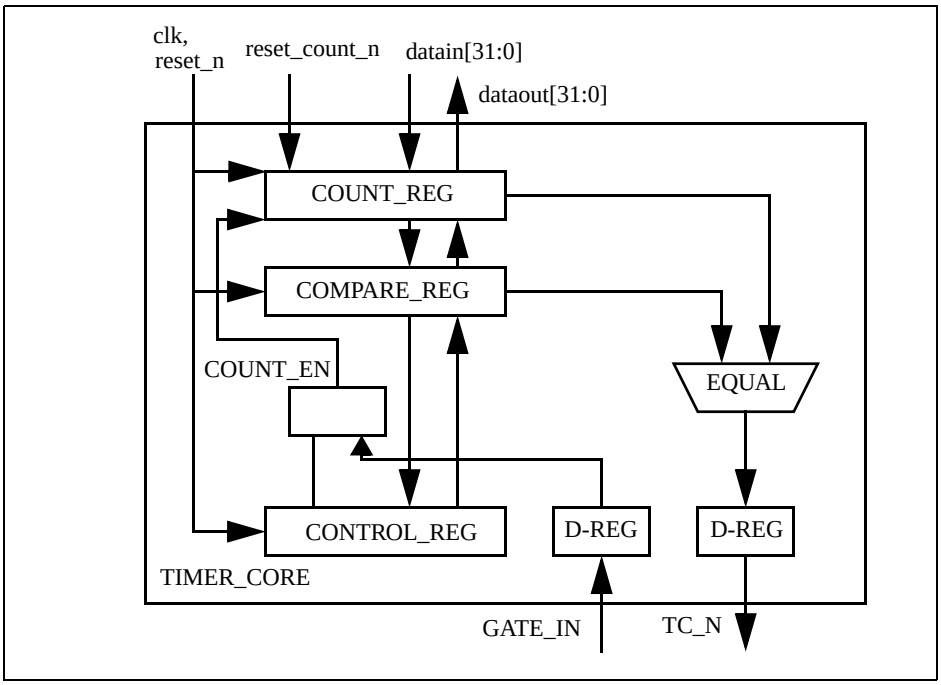


Figure 16.2 Diagram of Individual Timer Core

Overview

The general purpose timers, Timer 0 and Timer 1, can be used as a **real-time clock**. To meet real-time clock periods of 1 day or less, the timer 1 (internal signal timer_gate_n[1]) port is internally connected to the timer 0 (internal signal timer_tc_n[0]) port. This cascades the overflow count from Timer 0 into the effective clock for Timer 1, thus allowing a 64-bit count. General purpose Timer 2 can be used as a **Time-Slice clock**. The general purpose timers are organized as:

- *count register [31:0]*
- *compare register [31:0]*
- *control register[1] gate/timer bit*
- *control register[0] enable bit*

In addition to the general purpose Timer 0, Timer 1, and Timer 2, there are five separate dedicated timers for Watchdog, CPU bus time-out, IP bus time-out, SDRAM refresh, and WarmReset. WarmReset Timer 7 is used to count out clocks between the de-assertion of ColdReset_n and the de-assertion of Reset_n.

The timers are reset to `0x0000_0000` and count up to and equal to the value in their respective compare register. For the 1 clock of compare, tc_n is asserted. The output pin for timers 0 and 1 are synchronized (delayed) by one clock. At this point, the Count rolls over back to `0x0000_0000`. Note that Timer 7 is reset during a cold reset but not a warm reset.

Timers 0 and 1 contain an input gate mode, which uses the timer_gate_n pin as a clock enable for the timer ticks. The input is not synchronized (delayed) by the clock and feeds directly into the counter. To use the timer pins, the PIO Direction Register of the PIO Controller must first be programmed (for programming specifics of the PIO Direction Register, see Chapter 15). The default function is the timer_gate_n input pin. Timer pin functions are described in Table 16.1.

Timer 3 can only be used as a general purpose timer if the IP Bus Bridge Bus Error Control Register has the Watchdog Enable bit disabled. Timers 4 and 5 can only be used as general purpose timers if the IP Bus Bridge Bus Error Control Register has the CPU BusError and/or IP BusError Enable bits disabled, respectively.

Notes

Signal Definitions

Timer/Counter	Type	Alternative Signals	Descriptions
timer_tc_n[0]	I/O	pio[2], timer_gate_n[0]	Terminal count mode (timer_tc_n): Output indicating that the timer has reached its count compare value and has overflowed back to 0.
timer_gate_n[0]	I/O	timer_tc_n[0], pio[2]	Gate mode (timer_gate_n): input indicating that the timer may count one tick on the next clock edge.

Table 16.1 Pin Definitions for the Timer/Counter Signals

Register Mapping

The register sets for Timers 0 through 3 are mapped as listed in Table 16.2, Table 16.3 and Table 16.4. Register sets for the five timers dedicated to peripherals are mapped as listed in Table 16.5, Table 16.6, Table 16.7, Table 16.8, and Table 16.9.

Base Address Register 0	Register Name	Offset Address	Effective Address Register 0
1800_0700	Timer Control Register 0 (32 bits)	00	Base + Offset
	Timer Count Register 0	04	
	Timer Compare Register 0	08	

Table 16.2 Timer Register 0 (General Purpose) Address Map

Base Address Register 1	Register Name	Offset Address	Effective Address Register 1
1800_0710	Timer Control Register 1 (32 bits)	00	Base + Offset
	Timer Count Register1	04	
	Timer Compare Register 1	08	

Table 16.3 Timer Register 1 (General Purpose) Address Map

Base Address Register 2	Register Name	Offset Address	Effective Address Register 2
1800_0720	Timer Control Register 2 (32 bits)	00	Base + Offset
	Timer Count Register 2	04	
	Timer Compare Register 2	08	

Table 16.4 Timer Register 2 (General Purpose) Address Map

Base Address Register 3	Register Name	Offset Address	Effective Address Register 3
1800_0730	Timer Control Register 3 for Watchdog (32 bits)	00	Base + Offset
	Timer Count Register 3 for Watchdog	04	
	Timer Compare Register 3 for Watchdog	08	

Table 16.5 Register 3 for Watchdog Address Map

Notes

Base Address Register 4	Register Name	Offset Address	Effective Address Register 4
1800_0740	Timer Control Register 4 for CPU BusTimeout (BusError) (16-bits)	00	Base + Offset
	Timer Count Register 4 for CPU BusTimeout (BusError)	04	
	Timer Compare Register 4 for CPU BusTimeout (BusError)	08	

Table 16.6 Register 4 for CPU Bus Time-out Address Map

Base Address Register 5	Register Name	Offset Address	Effective Address Register 5
1800_0750	Timer Control Register 5 for IP BusTimeout (BusError) (16-bits)	00	Base + Offset
	Timer Count Register 5 for IP BusTimeout (BusError)	04	
	Timer Compare Register 5 for IP BusTime-out (BusError)	08	

Table 16.7 Register 5 for IP Bus Time-out Address Map

Base Address Register 6	Register Name	Offset Address	Effective Address Register 6
1800_0760	Timer Control Register 6 for DRAM Refresh (16-bits)	00	Base + Offset
	Timer Count Register 6 for DRAM Refresh	04	
	Timer Compare Register 6 for DRAM Refresh	08	

Table 16.8 Register 6 for DRAM Refresh Address Map

Base Address Register 7	Register Name	Offset Address	Effective Address Register 7
1800_0770	Timer Control Register 7 for Warm Reset (8-bits)	00	Base + Offset
	Timer Count Register 7 for Warm Reset	04	
	Timer Compare Register 7 for Warm Reset	08	

Table 16.9 Register 7 for Warm Reset Address Map

Timer Control Register Description

To meet real-time clock periods of 1 day or less, the internal signal timer_gate_n[1] port is internally connected to the internal signal timer_tc_n[0] port. (**Note**: on the RC32332, the signal timer_tc_n[0] is not present.) This cascades the overflow count from Timer 0 into the effective clock for Timer 1, thus allowing a 64-bit count. **Note**: the five dedicated peripheral timers are hardwired internally to their respective module.

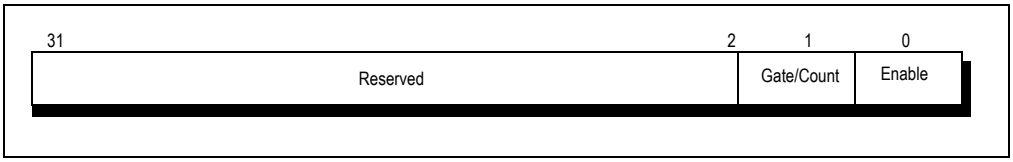


Figure 16.3 Timer Control Register Fields

Notes

Bit	Name	Description
1	Gate/Count	Note that the Gate option requires that the tc_n_gate_n pin first be set to the input direction. Value / Description: 1 – Gated Count 0 – Timer Count (default) Chapter 15 contains more programming specifics of the PIO Direction Register.
0	Enable	Enabled or disabled timers with this bit. Value / Description: 1 – Enabled (default) 0 – Disabled

Table 16.10 Timer Controller Register Field Descriptions

Timer Count Register

Timers are reset to 0x0000_0000 and count up to and equal to the value in their respective compare register. For the 1 clock of compare, Tc_n is asserted. The output pin for timers 0 and 1 are synchronized (delayed) by one clock. The Count then rolls back to 0x0000_0000.

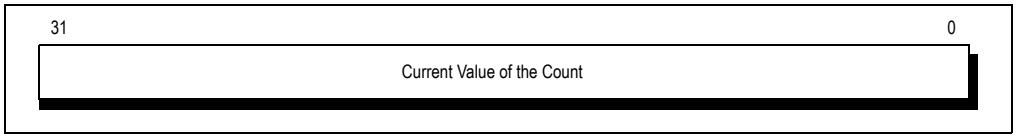


Figure 16.4 Count Register Fields

Bit	Name	Description
31:0	Timer Count	Value of 32-, 16- or 8- bit wide count

Table 16.11 Count Register Fields Descriptions

Timer Compare Register

Each of RC32334's eight timers count on each system clock, beginning from zero and time out after reaching a programmable compare value, resetting back to zero automatically.

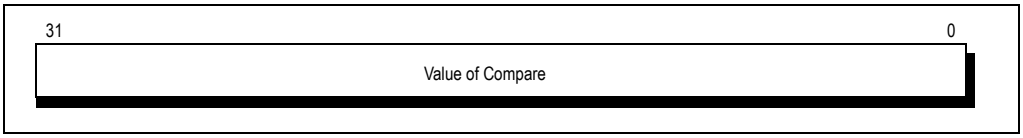


Figure 16.5 Compare Register Fields

Bit	Name	Description
31:0	Timer Compare	Value of 32-, 16- or 8-bit wide compare

Table 16.12 Compare Register Fields Descriptions

Notes

Timing Diagrams

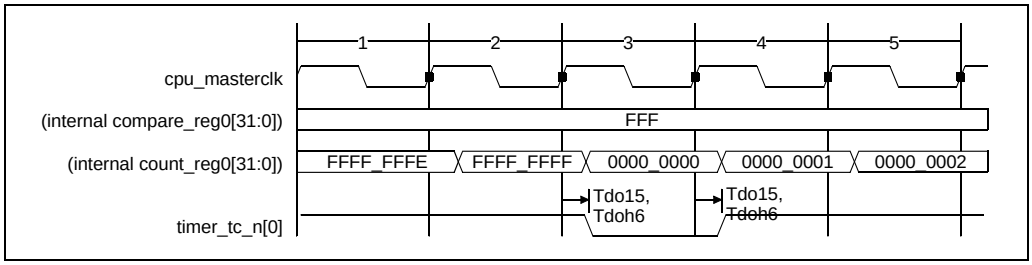


Figure 16.6 Timer Rollover Causing timer_tc_n to Toggle

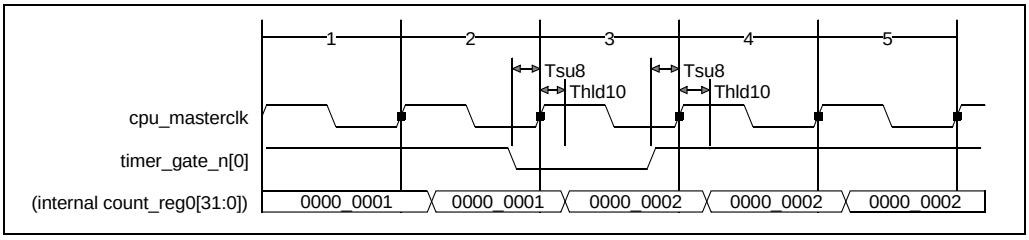


Figure 16.7 timer_gate_n Input Causing Timer to Count

UART Controller

Notes

Introduction

The two UARTs[1] in the RC32334 are 16550 compatible. The 16550 UART is an enhanced version of the 16450 UART. Functionally similar to the 16450, at power-up, the UARTs can be put into the 16550 mode, which then relieves the CPU core of software overhead. This allows execution of 16450 or 16550 compatible software. Two sets of 16-byte buffers are enabled during the 16550 mode: one set in the receive data path and one set in the transmit data path.

At any time during operation, the CPU core can read the UART status information, which includes the type and condition of the transfer operation as well as any error condition (parity, overrun, framing, or break interrupt). A baud rate generator is included that divides down the system clock by 1 to 65K. The baud rate generator provides the 16X clock for driving the transmitter and receiver logic. UART 0 is a full featured 16550 that supports the following features.

- *16-bit independent programmable Bit Rate Generator*
- *Baud rates from DC to 1.5M*
- *16 byte TX FIFO*
- *16 byte RX FIFO*
- *Programmable Data Format*
  - *5, 6, 7, or 8 Data Bit*
  - *Odd, Even or No Parity*
  - *1, 1 1/2 or 2 Stop Bits*
- *Modem control signal for channel 0*[2]
  - *RTS, CTS, DTR, DSR*
- *Maskable Interrupt Conditions*
- *Receive data available*
  - *Receive line status*
  - *Transmit holding register empty*
  - *Modem status*

UART 1[2] does not include flow control support, supporting only data transmit and receive pins. It is otherwise software compatible with UART 0. UART 0 registers begin at address location 0x18000800. UART 1 registers begin at address location 0x18000820.

Block Diagram

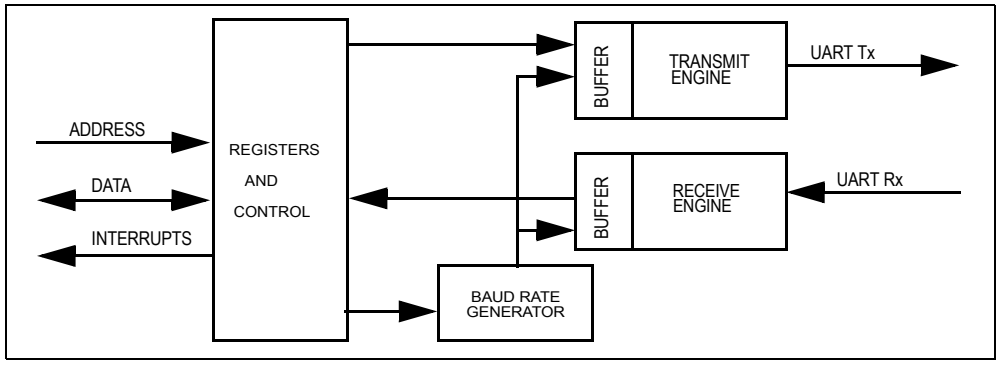


Figure 17.1 UART Block Diagram

1. There is only one UART in the RC32332.
2. Not in the RC32332.

Notes

Overview

The RC32334 provides two independent UARTs, each with a serial data transmit output and a serial data receive input. These UARTs each contain a 16-byte transmit buffer and a 16-byte receive buffer in the 16550 mode, a one-byte Transmit Holding Register and a one-byte Receive Holding Register. Data flow through the buffers only if enabled in the Buffer Control Register.

The user must set up the UART before operation. The transmit and receive parameters are set in the Line Control Register. The baud rate can also be set in the Divisor Latch Most and Divisor Latch Least Registers. The 16550 buffer mode may be enabled, if desired, in the Buffer Control Register, and should be chosen after reset is applied. Note that dynamically altering the buffer mode during a transmit or receive is not supported.

The UART contains a baud rate generator, and both the transmit and receive engines will run at the baud rate determined by the Divisor Latches. The Divisor Latches determine the baud rate by a two-byte divisor that divides down the RC32334 system clock. The divisor, in binary, loads into the Divisor Latch Least and Divisor Latch Most Registers.

A divisor value of one will disable the system clock divider, and the transmit and receive circuits will run at the system frequency. A divisor value of zero is modified to a divisor of 32 decimal (0020 hex) by the baud rate generator. To calculate the baud rate, use the following formula (the constant, 16, is used in the formula because the output frequency of the baud rate generator is 16 times the baud):

- *Baud rate = (system frequency) / (divisor * 16)*

Or, to calculate the divisor to load into the Divisor Latches, use the following formula:

- *Divisor = system frequency / (baud rate * 16)*

Example of a baud rate calculation: For a system frequency of 66 MHz and a baud rate of 9600 (values shown are decimal), calculate the divisor as follows:

Divisor = 66,000,000 / (9600 * 16) = 429.6875

Round off the ideal divisor to the nearest whole number, 430, and convert 430 to binary.

Load `0000_0001_1010_1110` into the Divisor Latches: `0000_0001` into the Most, and `1010_1110` into the Least. Some divisors and system frequencies will give a more accurate baud rate than others. Examples of other divisor values for typical baud rates are shown in Table 17.1.

To calculate the percent error of the divisor, use the following formula:

Percent error = ((difference of the whole divisor used and the ideal fractional divisor) / ideal fractional divisor) * 100.

Example of percent error calculation: ((430 - 429.6875) / 429.6875)) * 100 = 0.073% error.

System Frequency	Baud Rate	Divisor (decimal)
75MHz	9600	488
75MHz	75	65535
75MHz	1.5	3
66MHz	19200	214
66MHz	9600	430
66MHz	2400	1719
50MHz	9600	326
40MHz	9600	260
33MHz	9600	215
25MHz	9600	163

Table 17.1 Divisor Value Examples for Typical Baud Rates.

Notes

The user can employ two methods of transmitter empty and receive byte ready notification: interrupt driven or polling.

Also, by using the BCR DMA mode, the transmitter full and receive full conditions are available via the interrupt pending register in the Expansion Interrupt Controller described in Chapter 14.

UART Operation

To transmit a byte, the user writes a byte of data to the Transmit Holding Register. The UART controls inserting parity and the stop bit, then serially outputs the byte of data at the selected baud rate. When a byte of received data is ready for reading, the UART will notify the user with an interrupt, if enabled, through the Line Status Register. The byte of data is then read from the Receive Holding Register by the RC32300 CPU core. Receive errors are revealed to the user at the appropriate time (see the Line Status Register).

User Interrupts

In the RC32334 Interrupt Controller, there is one interrupt available to the user, which, unless masked, will activate the RC32334 interrupt pin to the CPU core (see Figure 17.2, Interrupt Flow). The Prioritized Interrupt is bit (0) in the IIR, it is inverted then passed to the RC32334 Interrupt Controller. It must be cleared in the UART first, then cleared in the RC32334 Interrupt Controller.

- ***Interrupt 0*** *- Prioritized Interrupt. Activated when one of the conditions in the IER is enabled. This is bit (0) in the IIR, inverted and sent to the Interrupt Controller. Masking it in the IER will prohibit it from being active in both the IIR and the Interrupt Controller. Masking it in the Interrupt Controller will still prohibit the interrupt from being active in the Interrupt Controller and downstream to the CPU; however, the IIR must still be cleared.*
- ***Interrupt 1*** *- Tx Rdy. Transmit Ready. See BCR DMA mode for full description.*
- ***Interrupt 2*** *- Rx Rdy. Receive Ready. See BCR DMA mode for full description.*

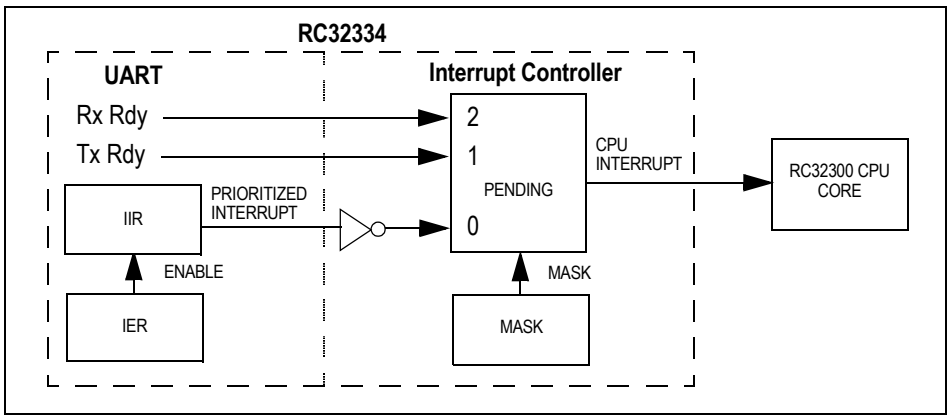


Figure 17.2 Interrupt Flow

Signal Definitions

Pin Name	Type	Description
UART Interface		
uart_rx[0]	I	**UART0** Serial data in
uart_tx[0]	O	**UART0** Serial data out
uart_dsr_n[0][1]	I	**UART0** Data Set Ready
uart_cts_n[0][1]	I	**UART0** Clear to Send
uart_rts_n[0][1]	O	**UART0** Request to Send

Table 17.2 RC32334 Pin Descriptions (Part 1 of 2)

Notes

Pin Name	Type	Description
uart_dtr_n[0][1]	O	**UART0** Data Terminal Ready
uart_rx[1][1]	I	**UART1** Serial data in
uart_tx[1][1]	O	**UART1** Serial data out

Table 17.2 RC32334 Pin Descriptions (Part 2 of 2)

1. Not in the RC32332.

UART 0&1 Registers

These registers enable UART functionality such as interrupt indication, data flow modes, and data receive/transmit formats. Some addresses are used more than once. To accomplish this, some register bits control register selection.

The RC32332 has only one serial port (UART0). All features in this user manual referencing UART1 should be ignored if the designer is planning to use the RC32332. Additionally, for UART0, all of the modem signals that were bonded out to the external pads in the RC32334—Request to Send (RTS), Clear to Send (CTS), Data Terminal Ready (DTR), and Data Set Ready (DSR)—are not accessible on the RC32332 pins. Therefore, the programming of these bits in the modem control registers does not perform any usable function in the RC32332.

UART 0 Registers

Address	Register Name	Descriptions
1800_0800	RBR/THR	Receiver Buffer Register/Transmitter Holding Register
1800_0804	IER	Interrupt Enable Register
1800_0800	DLL	Baud Divisor Latch, LS
1800_0804	DLM	Baud Divisor Latch, MS
1800_0808	IIR/BCR	Interrupt Identity Register/Buffer Control Register
1800_080C	LCR	Line Control Register
1800_0810	MCR	MODEM Control Register
1800_0814	LSR	Line Status Register
1800_0818	MSR	MODEM Status Register
1800_081C	SCR	Scratch Register
1800_0840	RR	Reset Register

Table 17.3 UART0 Register Address Map

UART 1 Registers[1]

Address	Register Name	Descriptions
1800_0820	RBR/THR	Receiver Buffer Register/Transmitter Holding Register
1800_0824	IER	Interrupt Enable Register
1800_0820	DLL	Baud Divisor Latch, 8 LSB
1800_0824	DLM	Baud Divisor Latch, 8 MSB
1800_0828	IIR/BCR	Interrupt Identity Register/Buffer Control Register

Table 17.4 UART1 Register Address Map (Part 1 of 2)

1. Not in the RC32332.

Notes

Address	Register Name	Descriptions
1800_082C	LCR	Line Control Register
1800_0830	MCR	MODEM Control Register
1800_0834	LSR	Line Status Register
1800_0838	MSR	MODEM Status Register
1800_083C	SCR	Scratch Register
1800_0860	RR	Reset Register

Table 17.4 UART1 Register Address Map (Part 2 of 2)

Receive Buffer Register (RBR)

This is a read-only register, accessed when the DLAB bit in the Line Control Register is set to zero. Bit 0 is the LSB and is the first bit serially received.

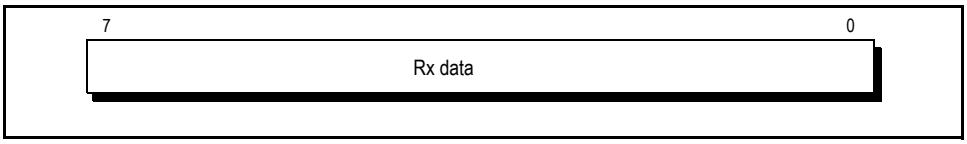


Figure 17.3 Receive Buffer Register

Transmit Buffer Register (TBR)

This is a write-only register, accessed when the DLAB bit in the Line Control Register is set to zero. Bit 0 is the LSB and is the first bit serially transmitted.

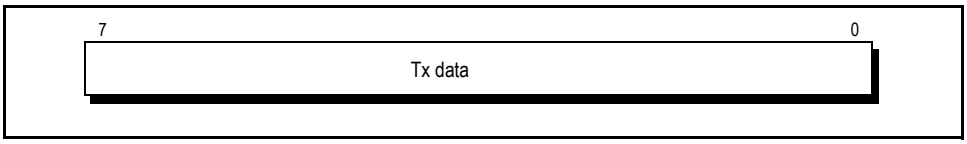


Figure 17.4 Transmit Buffer Register

Interrupt Enable Register (IER)

This is a read/write register, accessed when the DLAB bit in the LCR is set to zero. Disabling an interrupt in the IER prevents it from being indicated active in the IIR and from activating the interrupt signal to the Interrupt Controller.

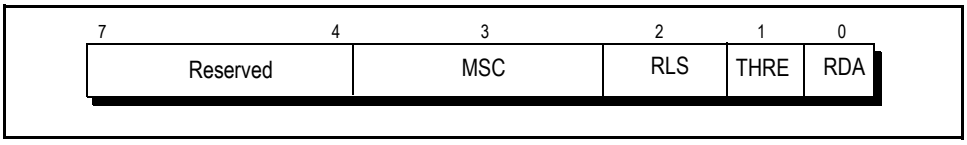


Figure 17.5 Interrupt Enable Register

Notes

Bit	Field Name	Description	Initial Value
7:4	Reserved		0x0
3	MSC	1 = Enable MODEM status change interrupt 0 = Disable interrupt	0x0
2	RLS	1 = Enable receiver line status interrupt 0 = Disable interrupt In the 16550 mode, enables character timeout interrupt.	0x0
1	THRE	1 = Enable transmitter holding register empty interrupt 0 = Disable interrupt	0x0
0	RDA	1 = Enable received data available interrupt 0 = Disable interrupt	0x0

Table 17.5 Interrupt Enable Register Field Descriptions

Divisor Latch Least Register (DLL)

This read/write register is accessed when the DLAB bit in the Line Control Register is set to one. Writing to the DLL or DLM will immediately change the baud rate. See Table 17.1 for additional baud rate information.

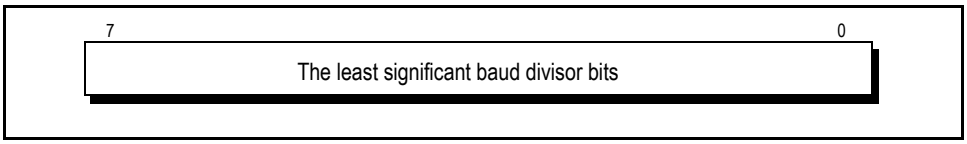


Figure 17.6 Divisor Latch Least Register (DLL)

Divisor Latch Most Register (DLM)

This read/write register is accessed when the DLAB bit in the Line Control Register is set to one. Writing to the DLL or DLM will immediately change the baud rate. See Table 17.1 for additional baud rate information.

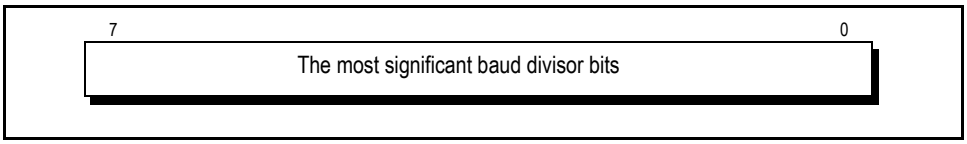


Figure 17.7 Divisor Latch Most Register (DLM)

Interrupt Identity Register (IIR)

This is a read-only register. The UART encodes four levels of priority and indicates the code in the IIR. When the software accesses the IIR, all interrupts are frozen and the highest pending interrupt is indicated in this register. The UART continues to record new interrupts while this access is taking place but does not change the contents of the IIR until the current access is complete.

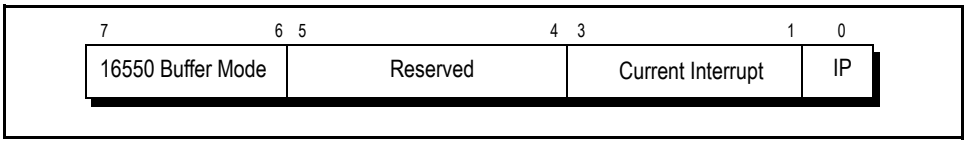


Figure 17.8 Interrupt Identity Register

Notes

<table>
<tr><th>Bit</th><th>Field Name</th><th>Description</th><th>Initial Value</th></tr>
<tr><td>7:6</td><td>16550 Buffer Mode</td><td>These two bits are set to ‘1’ when BCR(0) is set to ‘1’
<table>
<tr><th>Value</th><th>Status</th></tr>
<tr><td>1</td><td>Enable 16550 Buffer Mode</td></tr>
<tr><td>0</td><td>Disable 16550 Buffer Mode</td></tr>
</table></td><td>0x0</td></tr>
<tr><td>5:4</td><td>Reserved</td><td></td><td>0x0</td></tr>
<tr><td>3:1</td><td>Current Interrupt</td><td>A code describing the highest priority interrupt pending. Note that bit 3 is set to a ‘1’ in the 16550 Buffer Mode only.
<table>
<tr><th>Value</th><th>Status</th><th>Priority Level</th></tr>
<tr><td>000</td><td>Modem status</td><td>Fourth (lowest)</td></tr>
<tr><td>001</td><td>Transmitter Holding Register Empty
Writing to the THR will reset this interrupt</td><td>Third</td></tr>
<tr><td>010</td><td>Received Data Available
Rx data are available to read or the specified trigger level is reached. Either reading the RBR or if the buffer level drops below the trigger point resets the interrupt. The trigger level is specified in the BCR.</td><td>Second</td></tr>
<tr><td>011</td><td>Receiver Line Status
Occurs during an overrun error, parity error, framing error, or break interrupt. Reading the LSR resets the interrupt.</td><td>First (highest)</td></tr>
<tr><td>100</td><td>Reserved</td><td></td></tr>
<tr><td>101</td><td>Reserved</td><td></td></tr>
<tr><td>110</td><td>Character TimeOut Indication
No characters have been removed from or input to the receiver buffer during the last four character times and there is at least 1 character in it during this time. Buffer mode only.</td><td>Second</td></tr>
<tr><td>111</td><td>Receiver Line Status
Occurs during an overrun error, parity error, framing error, or break interrupt. Reading the LSR resets the interrupt.</td><td>First (highest)</td></tr>
</table></td><td>0x0</td></tr>
<tr><td>0</td><td>Interrupt Pending</td><td>This bit is inverted and mirrored in the RC32334 Interrupt Controller. This bit is active low in the IIR and active high in the Interrupt Controller.
<table>
<tr><th>Value</th><th>Status</th></tr>
<tr><td>1</td><td>No interrupt pending</td></tr>
<tr><td>0</td><td>Interrupt pending</td></tr>
</table></td><td>0x0
0x1</td></tr>
</table>

Table 17.6 Interrupt Identity Register Fields and Descriptions

Notes

Buffer Control Register (BCR)

The BCR register is a write-only register that enables and controls the use of the 16-byte receive and 16-byte transmit buffers.

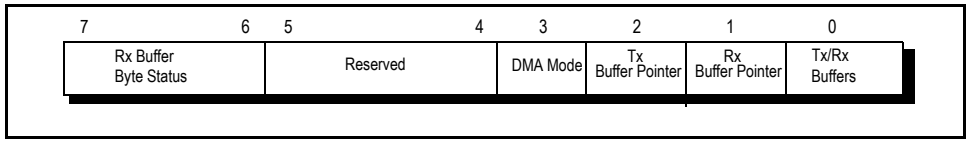


Figure 17.9 Buffer Control Register (BCR) Fields

Note: Changing from FIFO mode to non-FIFO mode does not automatically flush the FIFO. Switching FiFO modes dynamically while sending or receiving data is not supported.

Bit	Field Name	Description	Initial Value
7:6	Receive Buffer Byte Status	The receive buffer interrupt trigger level describes how many bytes are in the receive buffer. The setting of these bits affects the IIR. Value / Status: 11 — 14 bytes in the receive buffer 10 — 8 bytes in the receive buffer 01 — 4 bytes in the receive buffer 00 — 1 byte in the receive buffer	0x0
5:4	Reserved		0x0
3	DMA Mode	The TXRDY and RXRDY signals go to the Interrupt Controller, where they can act as an interrupt to the CPU. Masking these signals can only be accomplished in the Interrupt Controller Value / Status: 1 — TXRDY is activated when there are no bytes in the 16-byte buffer and is deactivated only when the buffer is full (16 bytes have been written to the buffer). RXRDY is activated when there are 16 bytes in the buffer and is deactivated only when the buffer is empty. 0 — TXRDY is activated when there are no bytes in the buffer and is deactivated when there is at least one byte in the buffer. RXRDY is activated when there is at least one byte in the buffer and is deactivated only when the buffer is empty.	0x0
2	Transmit Buffer Pointer	A one resets the transmit buffer pointer	0x0
1	Receive Buffer Pointer	A one resets the receive buffer pointer	0x0
0	Receive/Transmit Buffers	Enables the 16-byte transmit and receive buffers for 16550 mode operation. When switching between 16550 and 16450 modes, always reset the buffers.	0x1

Table 17.7 Buffer Control Register Field Descriptions

Notes

Line Control Register (LCR)

The LCR is a read/write register that controls the format of the data received and transmitted.

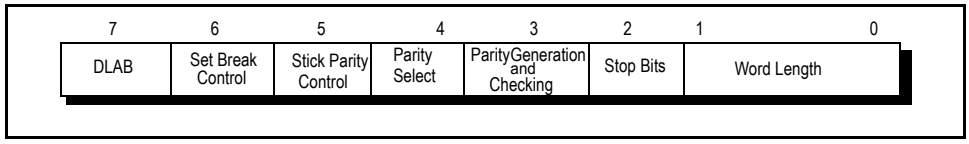


Figure 17.10 Line Control Register Fields

Bit	Field Name	Description	Initial Value
7	Divisor Latch Access Bit	0 = RBR,THR and IER registers selected 1 = DLL and DLM registers selected	0x0
6	Set Break Control	0 = Normal transmit data 1 = Transmit data will be forced to a zero (spacing state), causing a break condition to be transmitted	0x0
5	Stick Parity Control	**Bit / Value / Status** 5:3 / 111 / Parity bit transmitted and checked as a zero 5:3 / 101 / Parity is transmitted and checked as a one 5 / 0 / Stick parity disabled	0x0
4	Parity Select	1 = Even parity 0 = Odd parity	0x0
3	Parity Generation and Checking	1 = Enable parity generation and checking 0 = Disable parity generation and checking	0x0
2	Stop Bits	Controls the number of stop bits generated **Value / Status** 0 / One stop bit generated 1 / 5-bit word length: 1 1/2 stop bits generated 6, 7 or 8-bit word length: 2 stop bits generated	0x0
1:0	Word Length	**Value / Status** 00 / 5-bits 01 / 6-bits 10 / 7-bits 11 / 8-bits	0x0

Table 17.8 Line Control Register Field Descriptions

Modem Control Register (MCR)

This is a read/write register that controls the MODEM operation. For UART0, the Data Terminal Ready (DTR) and Request to Send (RTS) bits in the modem status register for this UART are muxed with PIO pins of the RC32334 device. For UART1, none of the modem signals in the related modem status register are connected to the external pins and, therefore, do not perform any usable function.[1]

1. For UART0 in the RC32332, no modem signals are connected to external pins and therefore do not perform any usable function. Also, UART1 is not present in the RC32332.

Notes

Refer to Chapter 15, “Programmable I/O (PIO) Controller,” to configure these signals to be enabled externally.

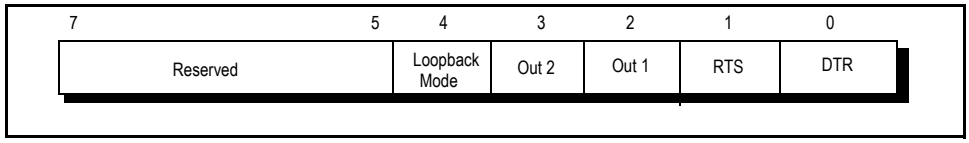


Figure 17.11 MODEM Control Register Fields

Bit	Field Name	Description	Initial Value
7:5	Reserved		0x0
4	Loopback mode	1 = Enable loopback mode for diagnostic testing of the UART In this mode, the transmit data pin is internally directed to the receiver logic, replacing the receive data input of the CPU 0 = Disable loopback	0x0
3	Out 2	No connection to pin	0x0
2	Out 1	No connection to pin	0x0
1	Request to Send	Connected to pin B1, uart_rts_n[0]	0x0
0	Data Terminal Ready	Connected to pin C3, uart_dtr_n[0]	0x0

Table 17.9 MODEM Control Register Field Descriptions

Line Status Register (LSR)

The LSR is a read/write register that provides UART status information to software.

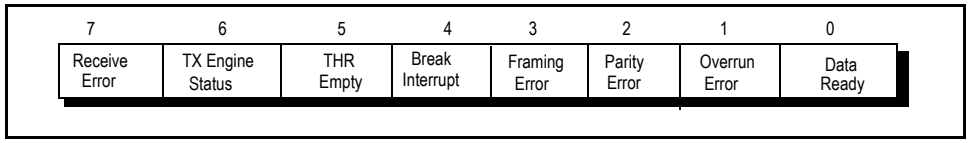


Figure 17.12 Line Status Register Fields

Bit	Field Name	Description	Initial Value
7	Receive Error	1 = Receive error detected on a character in the buffer A receive error could be parity, framing, or break conditions 0 = No receive error	0x0
6	Transmit Engine Status	1 = Transmit engine is not active This means that the transmit buffer and the THR are both empty 0 = Transmit transfer in progress	0x1
5	THR Empty	1 = Transmit Holding Register is empty 0 = THR not empty	0x1
4	Break Interrupt	1 = Receive data is at the spacing (zero) level for more than a full word transmission time The bit is reset when software reads the LSR. The error is registered in the LSR when the associated character is at the top of the buffer in the 16550 buffer mode 0 = No break interrupt	0x0

Table 17.10 Line Status Register Field Descriptions (Part 1 of 2)

Notes

Bit	Field Name	Description	Initial Value
3	Framing Error	1 = Received data did not have a valid stop bit The bit is reset when software reads the LSR. The error is registered in the LSR when the associated character is at the top of the buffer in the 16550 buffer mode 0 = No framing error	0x0
2	Parity Error	1 = Parity indication of the received data does not match the configuration in the LCR The bit is reset when software reads the LSR. The error is registered in the LSR when the associated character is at the top of the buffer in the 16550 buffer mode 0 = No parity error	0x0
1	Overrun Error	1 = The data in the receive buffer were overwritten before the CPU read them This error is registered in the LSR as it occurs 0 = No overrun error	0x0
0	Data Ready	1 = At least one byte of data is ready to read 0 = No data in THR or receive buffer	0x0

Table 17.10 Line Status Register Field Descriptions (Part 2 of 2)

Modem Status Register (MSR)

The MSR is a read/write register that controls MODEM operation. For UART0, the Data Set Ready (DSR) and Clear to Send (CTS) bits in the modem status register for this UART are connected to the external pins of the RC32334 device. For UART1, none of the modem signals in the related modem status register are connected to the external pins and, therefore, do not perform any usable function.[1]

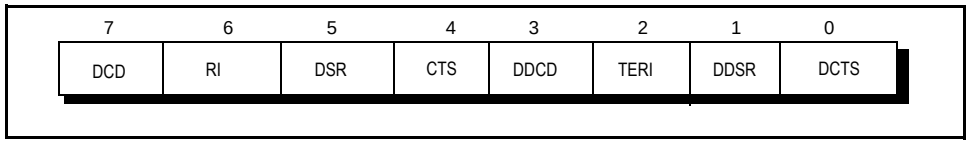


Figure 17.13 MODEM Status Register Fields

Bit	Field Name	Description
7	Data Carrier Detect	No connection to pins
6	Ring Indicator	No connection to pins
5	Data Set Ready	Connected to pin B2, uart_dsr_n[0]
4	Clear to Send	Connected to pin A1, uart_cts_n[0]
3	Delta Data Carrier Detect	No connection to pins
2	Trailing edge of ring indicator	No connection to pins
1	Delta Data Set Ready	No connection to pins
0	Delta Clear to Send	No connection to pins

Table 17.11 MODEM Status Register Field Descriptions

1. For UART0 in the RC32332, no modem signals are connected to external pins and therefore do not perform any usable function. Also, UART1 is not present in the RC32332.

Notes

Scratch Register (SCR)

The SCR is a read/write register that can be used as temporary storage by the user and has no affect on the UART operation.

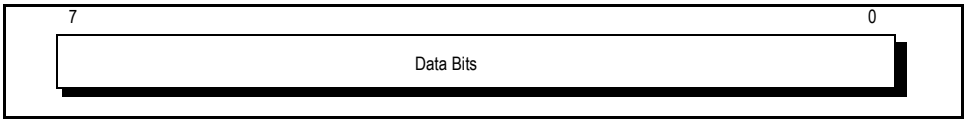


Figure 17.14 Scratch Register Field

Bit	Field Name	Description	Initial Value
7:0	Data Bits	This register has no effect on UART operation and can be used as temporary storage	0x0

Table 17.12 Scratch Register Field Descriptions

Reset Register (RR)

Writing any data value to this register will reset the UART channel.

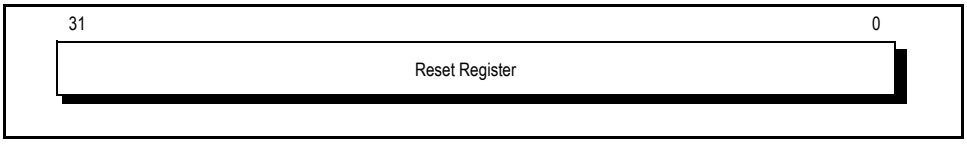


Figure 17.15 Reset Register Field

Timing Diagram

Timing of the UART inputs and outputs are shown in Figure 17.16. Note that the UART data setup/hold protocol itself implies asynchronous timing. uart_rx[0] and uart_tx[0] are shown in an input and output respectively. The other UART signals, including uart_rx[1:0], uart_tx[1:0], uart_dsr_n[0], uart_cts_n[0], uart_rts_n[0], and uart_dtr_n[0] have similar timing in their input and output modes.

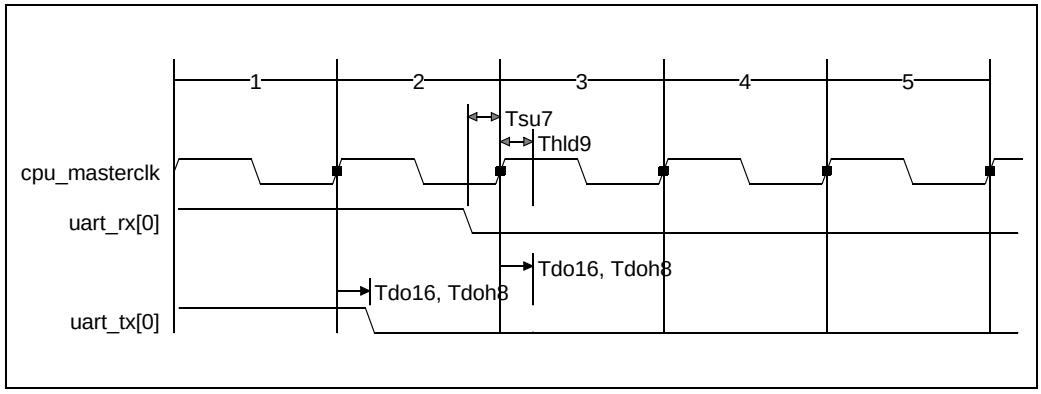


Figure 17.16 UART Timing

Serial Peripheral Interface

Notes

Introduction

The RC32334 supports the Serial Peripheral Interface (SPI) master capability, to provide an interface to low-cost serial peripherals. This interface uses four pins: serial data in (spi_miso), serial data out (spi_mosi), serial clock (spi_sck) and slave chip select (spi_ss_n), as shown in Figure 18.1. This serial interface includes an 8-bit shift register, a system clock divider, a SCK generator, 4 registers, and a state machine. The SPI interface provides the following features and capabilities:

- *Full-Duplex Operation*
- *Master Modes only*
- *System Clock to SPI Clock divider/prescalar*
- *Four Programmable Master Mode Frequencies*
- *Serial Clock with Programmable Polarity and Phase*
- *Write Collision Error Flag*

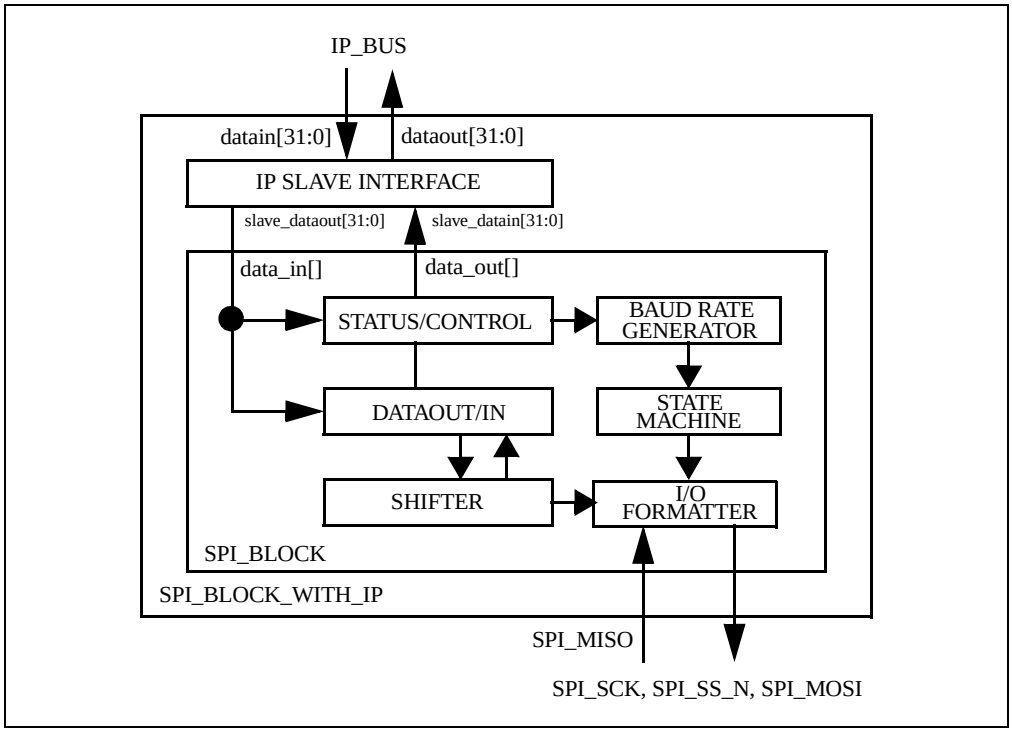


Figure 18.1 SPI Block Diagram

The master SPI allows fully duplexed, synchronous serial communication between the RC32334 and other peripheral devices, such as an ATMEL SPI or Serial E2PROMs. When an SPI transfer occurs, an 8-bit data is shifted out of spi_mosi, simultaneously as an 8-bit data is shifted into spi_miso.

When a master device transmits data to a slave device via the spi_mosi line, the slave device responds by sending data to the master device via the master's spi_miso line. This implies full duplex transmission, with both data out and data in synchronized with the same clock signal. Thus, the byte transmitted is replaced by the byte received and eliminates the need for separate transmit-empty and receiver-full status bits. A single status bit (SPIF) is used to signify that the I/O operation has been completed.

Notes

The SPI is double-buffered on read, but not on write. If a write is performed during data transfer, the transfer occurs uninterrupted, and the write will be unsuccessful. This condition will cause the write collision (WCOL) status bit in the SPSR to be set. After a data byte is shifted, the SPIF flag of the SPSR is set.

The spi_sck pin is an output pin that idles high or low, depending on the CPOL bit in the SPCR, until data is written to the shift register, at which point eight clocks are generated to shift the 8 bits of data, and then spi_sck goes idle again.

Signal Descriptions

Signal Name	Type	Alternate Signal Name	Description
spi_mosi	I/O	pio[10]	**SPI Data Output** Serial mode: Output pin from RC32334 as an Input to a Serial Chip for the Serial data input stream. In PCI satellite mode, acts as an Output pin from RC32334 that connects as an Input to a Serial Chip for the Serial data input stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[10]. 2nd Alternate function: pci_eeprom_mdo.
spi_miso	I/O	pio[7]	**SPI Data Input** Serial mode: Input pin to RC32334 from the Output of a Serial Chip for the Serial data output stream. In PCI satellite mode, acts as an Input pin from RC32334 that connects as an output to a Serial Chip for the Serial data output stream for loading PCI Configuration Registers in the RC32334 Reset Initialization Vector PCI boot mode. Defaults to input direction at reset time. 1st Alternate function: PIO[7]. 2nd Alternate function: pci_eeprom_mdi.
spi_sck	I/O	pio[9]	**SPI Clock** Serial mode: Output pin for Serial Clock. In PCI satellite mode, acts as an Output pin for Serial Clock for loading PCI Configuration Registers in the RC323334 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[9]. 2nd Alternate function: pci_eeprom_sk.
spi_ss_n	I/O	pio[8]	**SPI Chip Select** Output pin selecting the serial protocol device as opposed to the PCI satellite mode EEPROM device. Alternate function: PIO[8]. Defaults to the output direction at reset time.

Table 18.1 SPI Signal Descriptions

The RC32334's SPI module initiates a transmission by writing to the SPI data register (SPDR), which moves the data to a shift register and transmission immediately begins. After eight serial clock cycles, the SPI sets the SPI flag (SPIF) and transmission ends.

Before the SPI begins another transmission, SPIF must be cleared by reading the SPI Status Register and then the SPDR. Interrupts are generated at the end of a transmission, if the SPI Interrupt Enable Bit has been set. Figure 18.4 lists and describes the SPI control register fields. Serial clock polarity and phase. The RC32334's SPI controller does not implement the SPI slave mode or SPI multimastering.

Notes

Most peripherals require a multibyte command sequence. For multibyte commands, the spi_ss_n pin must be programmed via the general purpose PIO output data mode to remain asserted through the multiple bytes. To accommodate the various serial communication requirements of peripheral devices, software can change the phase and polarity of the SPI serial clock.

The clock polarity bit (CPOL) and the clock phase bit (CPHA), both in the SPCR, control the timing relationship between the serial clock and the transmitted data. Most typical peripherals use either the (0,0) mode or the (1,1) mode, where (CPOL CPHA) indicate the mode

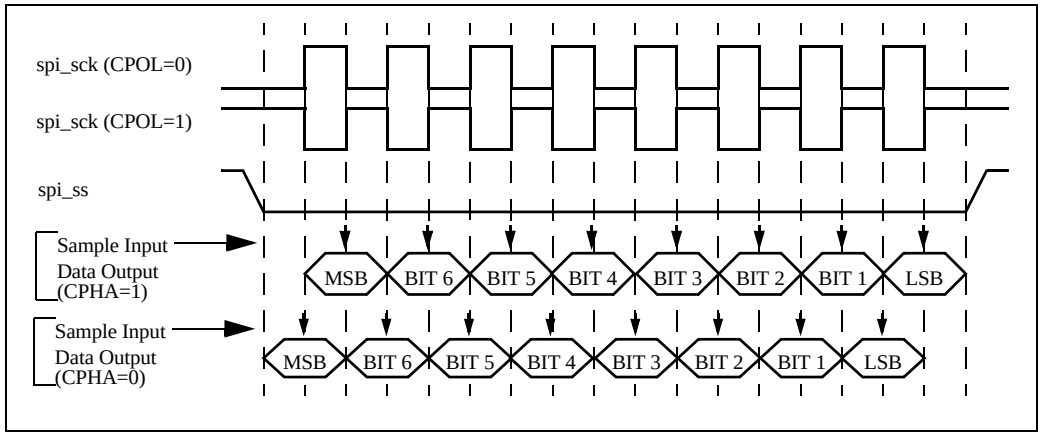


Figure 18.2 Serial Peripheral Interface (SPI) Clock/Data Timing

SPI Data Setup/Hold and Delay Timing

The SPI protocol specifies its data input and output timing relative to spi_sck transitions. However, in reality, the RC32334 SPI channel accepts input and delivers output data, based on the cpu_masterclk rising edge, immediately after a spi_sck transition. Thus, if the SPI setup and hold time is met relative to spi_sck—since spi_sck is much slower than cpu_masterclk—the setup and hold to the spi_sck enabled cpu_masterclk input latch will also be met. Similarly, if the SPI slave device latches data with spi_sck, since spi_sck is much slower than cpu_masterclk, the setup (and hold) to the slave is also met.

SPI Setup and Register Descriptions

Notes

The following describes the typical setup of SPI, which occurs during boot time:

1. The SPI shares data and clock pins with the PCI EEPROM if the RC32334 is booted in the boot-from-PCI reset mode. An internal PCI EEPROM Busy Flag in the PCI Controller is used by the PCI Controller to determine if the PCI is finished loading its data from the PCI EEPROM. The RC32334 internal PCI EEPROM Busy Flag automatically switches the pin effect usage to SPI in all boot reset modes by the time of the first instruction fetch after a CPU reset.

   Note: Even if the PCI is not used, SPI usage still requires the PCI to assert pci_rst_n in order to properly set the internal PCI EEPROM Busy Flag.
2. SPI signal functions are routed via the PIO Controller, so the PIO Controller will generally be initialized to the Effect Mode, with corresponding direction for each SPI pin. At reset time, the default Effect Mode and Direction are set up for the PCI EEPROM and also for SPI.
3. The SPI Clock Register, SPCNT, is written.
4. The SPI Control Register, SPCNTL, including the SPE Enable Bit is written.
5. The data being sent to the SPI Slave are written to the SPI Data Register (SPDR).
6. The SPI Controller will initiate the hardware protocol on the SPI pins. The protocol has the Master receive data from the Slave at the same time the Master sends data to the Slave.
7. Wait either for:
   - *SPI Interrupt. After receiving an SPI Interrupt via the Interrupt Controller, read the SPI Status Register SPIF and MODF Flags.*
   - *Poll the SPI Status Register SPIF and MODF Flags.*
8. If the SPIF Flag is set, indicating the transaction is complete, reading the SPI Status Register resets the SPIF Flag.
9. Read the data from the SPI Data Register.
10. Repeat Steps 5 through 10, as needed.

SPI Interrupt Description

SPI produces a single interrupt. Before feeding into the Interrupt Controller, the interrupt is enabled/disabled via the SPIE Interrupt Enable Bit in the SPI Control Register (SPCNTL). SPI asserts the interrupt line if either the SPIF and/or the MODF bit of the SPI Status Register is set, indicating an unusual Slave (mis-)operation or that the present transaction is complete.

By disabling SPI, via the SPI Control Register SPE Bit, the SPI signal pins may be reused for other functions, including general purpose Programmable I/O pins, or for bit-blasting the PCI EEPROM after PCI initialization; for example, to write to the PCI EEPROM.

Base Address	Register	Offset Address	Effective Address
1800_0900	Serial Peripheral Clock Divisor/Prescalar Register (SPCNT)	00	Base + Offset
	Serial Peripheral Control Register (SPCNTL)	04	
	Serial Peripheral Status Register (SPSR)	08	
	Serial Peripheral Data I/O Register (SPDR)	0C	

Table 18.2 SPI Register Address Map

Serial Peripheral Clock Register (SPCNT)

The SPCNT register is used to program the divide-down clock count prescalar, which then goes to the basic SPI clock divisor controlled by the SPCNTL register SPR field. The SPCNT register is used as a compare value to count the number of system clocks (cpu_masterclks) per 0.5 SPI divide-down/prescalar clock. The default is 0x00.

spi_sck = system clock / [2*(SPCNT+1) * SPR]

Notes

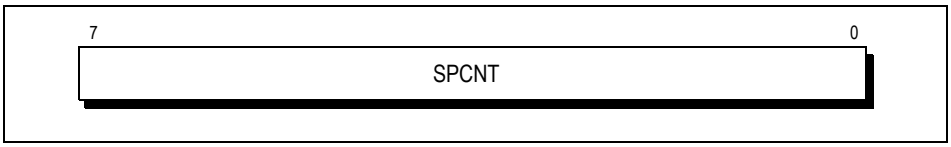


Figure 18.3 SPI Clock Register Field

Bits	Field	Function
7:0	SPCNT	Used to divide the system clock to the 1-4 MHz input clock rate required for SPI.

Table 18.3 SPI Clock Register (SPCNT) Field Description

Serial Peripheral Control Register (SPCNTL)

SPI enables features and interrupts through the Serial Peripheral Control Register, i.e., the slave mode rates, clock phase and polarity, master/slave state, as listed in Table 18.4. Fields of the SPCNTL register are shown in Figure 18.4. The default is 0x10.

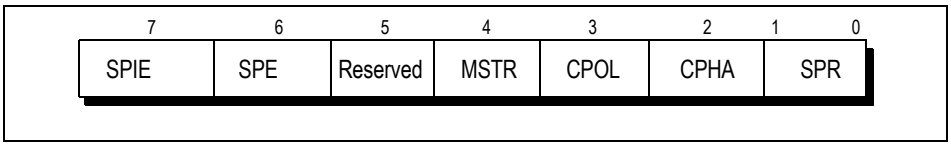


Figure 18.4 Serial Peripheral Control (SPCNTL) Register Fields

Bits	Field	Description
7	SPIE	Interrupt Enable Value 0: SPI interrupts are disabled (default) Value 1: SPI interrupts are enabled if SPIF is set to one
6	SPE	System On/Off Value 0: SPI system is off (default) Value 1: SPI system is on
5	Reserved	
4	MSTR	Master/Slave Mode Value 0: Reserved Value 1: SPI is configured as a master (default)
3	CPOL	Clock polarity. When the clock polarity bit is cleared and data is not being transferred, a steady state low value is produced at the SCK pin of the master device. Conversely, if this bit is set, the SCK pin will idle high. This bit is also used in conjunction with the clock phase control bit to produce the desired clock-data relationship between master and slave. Value 0: spi_sck pin at logic zero between transmissions (default) Value 1: spi_sck pin at logic one between transmissions

Table 18.4 SPI Control Register Field Descriptions (Part 1 of 2)

Notes

Bits	Field	Description
2	CPHA	The clock phase bit, in conjunction with CPOL bit, controls the clock-data relationship between master and slave. The CPOL bit can be thought of as simply inserting an inverter in series with the spi_sck line. The CPHA bit selects one of two fundamentally different clocking protocols. **Value** / **Description** 0 – First edge on spi_sck latches data (default) 1 – Edge following first edge on spi_sck latches data
1:0	SPR	These two bits select one of the following four bit rates when RC32334 is a master. These bits have no effect when in slave mode. **Value** / **Description** 00 – Divided by 2 (default) 01 – Divided by 4 10 – Divided by 16 11 – Divided by 32

Table 18.4 SPI Control Register Field Descriptions (Part 2 of 2)

Serial Peripheral Status Register (SPSR)

Note that during a transfer, writing to the SPDR register (see Table 18.6) causes a write collision error and sets the WCOL bit of the status register. Clear the WCOL bit by reading the status register (SPSR) with the WCOL bit set, and then reading or writing the SPI data I/O register. The default is 0x80.

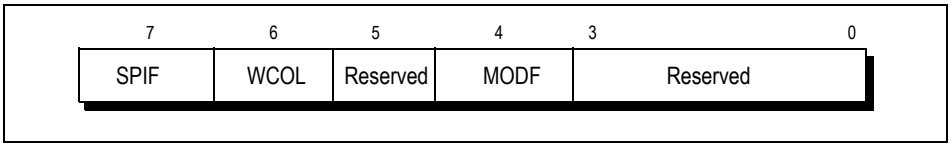


Figure 18.5 SPI Status Register (SPSR) Fields

Bits	Field	Description
7	SPIF	SPI Transfer Complete Flag The serial peripheral data transfer flag bit is set upon completion of data transfer between the processor and external device. If SPIF goes high, and the SPIE is set, a serial peripheral interrupt is generated. Clearing the SPIF bit is accomplished by reading the SPSR (with SPIF set), followed by an access of the SPDR. Unless SPSR is read (with SPIF set) first, attempts to write to SPDR are inhibited. **Value** / **Description** 0 – Ready to transfer 1 – Transfer complete. SPDR writes inhibited.

Table 18.5 SPI Status Register (SPSR) Field Descriptions

Notes

Bits	Field	Description
6	WCOL	Write Collision The write collision bit is set when an attempt is made to write to the serial peripheral data register while data transfer is taking place. Clearing the WCOL bit is accomplished by reading the SPSR (with WCOL set), followed by an access to SPDR. Value: 0 — Normal Value: 1 — An attempt to write to SPI while a transfer was in progress. The write is ignored.
5	Reserved	
4	MODF	Master Error Flag Asserts an error condition if a write to the SPDR is done while the SPI interface is in non-master (slave) mode. Refer to the MSTR field in Table 18.4 for more information on the master mode. Clearing the MODF bit is accomplished by optionally reading the SPSR first, followed by a srite of "1" to the SPCR MSTR field. Value: 0 — Normal Value: 1 — This is an error condition.
3:0	Reserved	

Table 18.5 SPI Status Register (SPSR) Field Descriptions

Serial Peripheral Data I/O Register (SPDR)

The serial peripheral data I/O register is used to transmit and receive data on the serial bus. Only a write to this register will initiate transmission/reception of another byte, and this will only occur in the master device. At the completion of transmitting a byte of data, the SPIF status bit is set in both the master and slave devices.

When the user reads the serial peripheral data I/O register, a buffer is actually being read. The first SPIF must be cleared by the time a second transfer of data from the shift register to the read buffer is initiated or an overrun condition will exist. In cases of overrun, the byte that causes the overrun is lost.

A write to the serial peripheral data I/O register is not buffered and places data directly into the shift register for transmission.

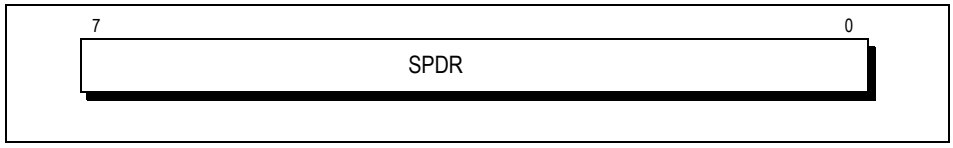


Figure 18.6 SPI Data I/O Register

Bits	Field	Description
7:0	SPDR	SPI Data I/O Register A write to this register with the transmit data value automatically initiates simultaneous transmission and reception of data. At the completion of data transfer, the SPIF status bit is set and the SPSR register is read. Then the receive data can be read from this register.

Table 18.6 SPI Data I/O Register (SPDR) Field Description

Notes

Interface to SPI Serial E2PROMs by ATMEL (AT25128)

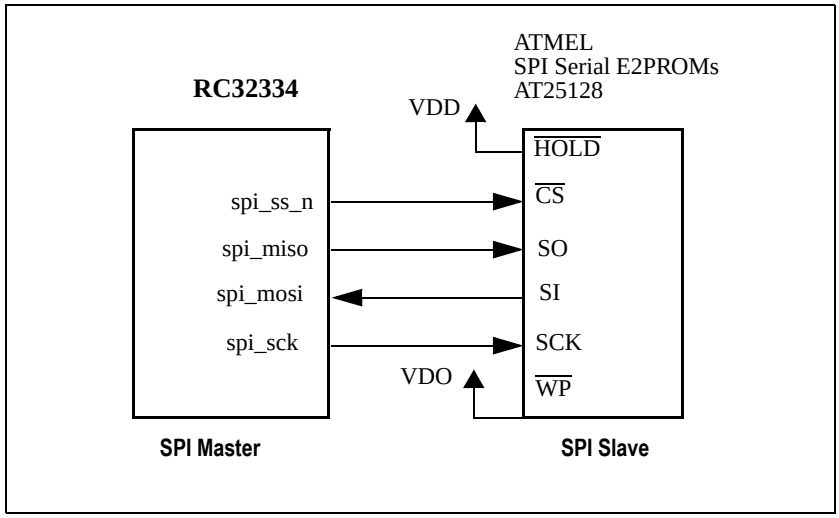


Figure 18.7 Illustration of Glueless Connection Between RC32334 Processor and ATMEL SPI Serial E2PROMs

Master Programming Example

The following sequence initializes the SPI to run at 2MHz (for this example, assume the system clock is running at 67 MHz).

1. Write SPCNT register with 0x0000_0008. This will divide the internal SPI clock down to 3.7MHz (67/ [(8+1)*2]).
2. Write SPCR register with 0x0000_00f0. SPIE = 1, SPI interrupt enabled. SPE=1, enable SPI for transmission. MSTR=1, this bit is always programmed to 1, since RC32334 SPI only supports master mode. Use (0,0) mode where CPOL=0, clock is low when SPI is not active. CPHA=0, latch data on the first active edge. SPR[1:0]=0, divide internal clock by 2 to generate SPI clock (3.7 / 2 = 1.85MHz).
3. Write Interrupt Mask Register 14 (0x1800_05e4) with 0x0000_0001. Enable SPI-generated system interrupt.
4. Read SPSR and SPDR to clear SPIF bit.
5. Assert spi_ss_n by using the spi_ss_n pin in its general purpose PIO output mode and setting its data low.
6. Write SPDR with the value to transmit to start the SPI transmission.
7. Wait until the SPI interrupt occurs, the interrupt routine will perform the following step:
8. Read SPSR. Make sure there is no error condition.
9. Read SPDR. Get the receive value from SPDR, which clears the SPIF bit in the SPSR register.
10. Write SPI Interrupt Clear Register (0x1800_05e8) with 0x0000_0001. Clear SPI Interrupt Pending Register.
11. If finished with a (multi-) byte command sequence (i.e., a read command/address byte 1/address byte 2/data 4-byte sequence), then de-assert the spi_ss_n pin via the PIO data register.
12. Repeat steps 5 - 11 as needed.

Timing Diagrams

Timing of the SPI Clock-to-Data Output Relationship is shown in Figure 18.8, and timing of the relationship of clock-to-data input is shown in Figure 18.9. Note that the SPI data setup/hold protocol itself implies 0.5 spi_sck clock setup/hold relative to the master and slave devices.

Notes

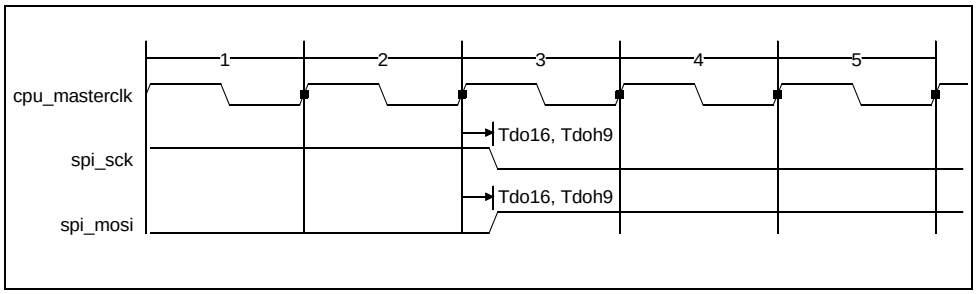


Figure 18.8 SPI Clock-to-Data Output Relationship

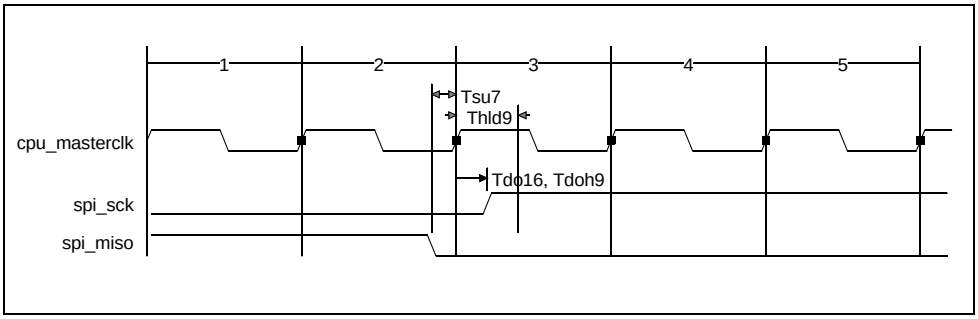


Figure 18.9 SPI Clock-to-Data Input Relationship

Notes

Clocking, Reset, and Initialization

Notes

Introduction

This chapter provides a description of the clock signals (“clocks”) that are used on the RC32334 processor. For a discussion of the basic system clocks and system timing parameters, see Chapter 8.

Signal Terminology

In this chapter and throughout the manual, when describing signal transitions, the following terminology is used:

- *Rising edge indicates a low-to-high (0 to 1) transition.*
- *Falling edge indicates a high-to-low (1 to 0) transition.*
- *Clock-to-Q delay is the amount of time it takes for a signal to move from the input of a device (clock) to the output of the device (Q).*

These terms are illustrated in Figure 19.1 and Figure 19.2.

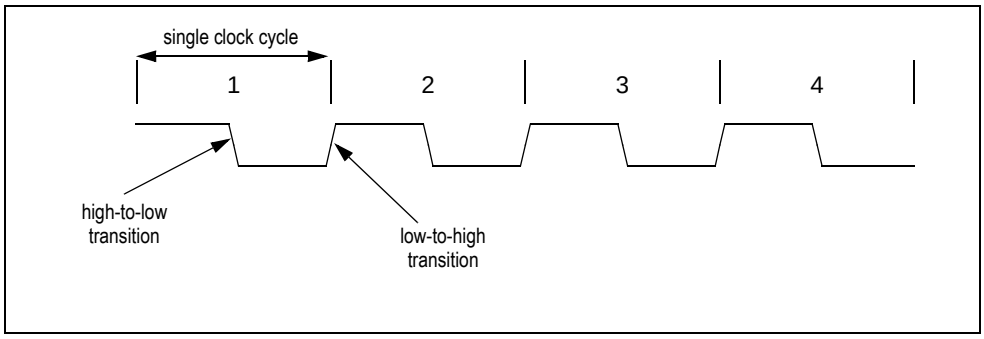


Figure 19.1 Signal Transitions

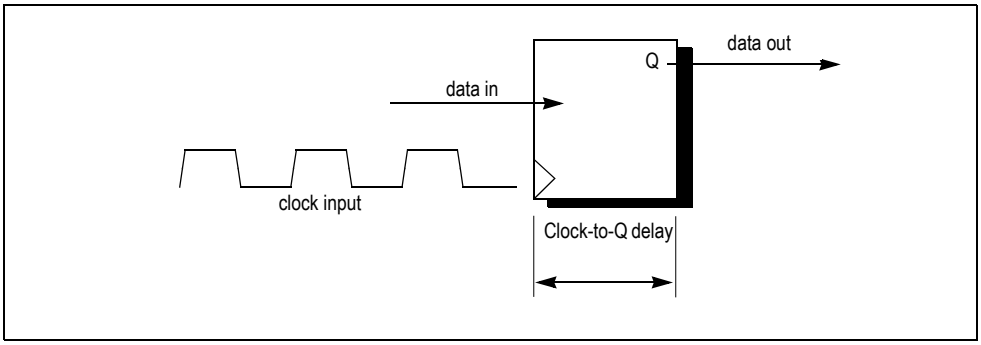


Figure 19.2 Clock-to-Q Delay

Basic System Clocks

The RC32334 processor has a single input clock, cpu_masterclk.

Cpu_masterclk

The cpu_masterclk input must meet the maximum rise time (T_{MCRise}), maximum fall time (T_{MCfall}), minimum clock high ($T_{MCkHigh}$) time, minimum clock low (T_{MCkLow}) time, and input jitter ($T_{JitterIn}$) parameters for proper phase locked loop (PLL) operation.

Notes

The processor bases all internal clocking on the single cpu_masterclk (MClk) input signal. The RC32334 uses cpu_masterclk to sample data at the system interface and to clock data into the processor system interface output register.

The external agent should use cpu_masterclk for the global system clock and for clocking the output registers of an external agent. Figure 19.3 shows the input, output and hold time parameters measured at the midpoint of the rising clock edge.

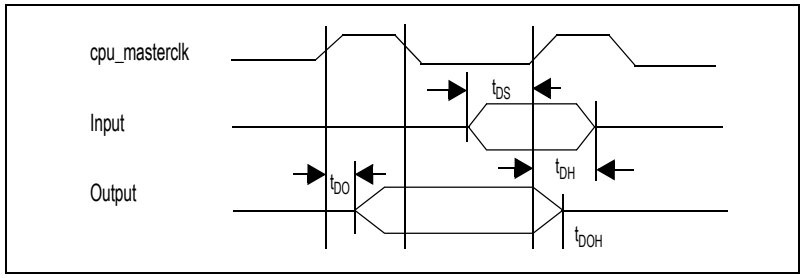


Figure 19.3 System Clocks Data Setup, Output, and Hold Timing

PClock

By multiplying cpu_masterclk 2, 3, or 4 times (programmed during the reset or initialization sequence through the Clock Multiplier configuration mode bits), the processor generates the internal pipeline clock rate, PClock, which is used by all internal registers and latches.

Figure 19.4 shows the clocks for a cpu_masterclk-to-PClock multiply by 2.

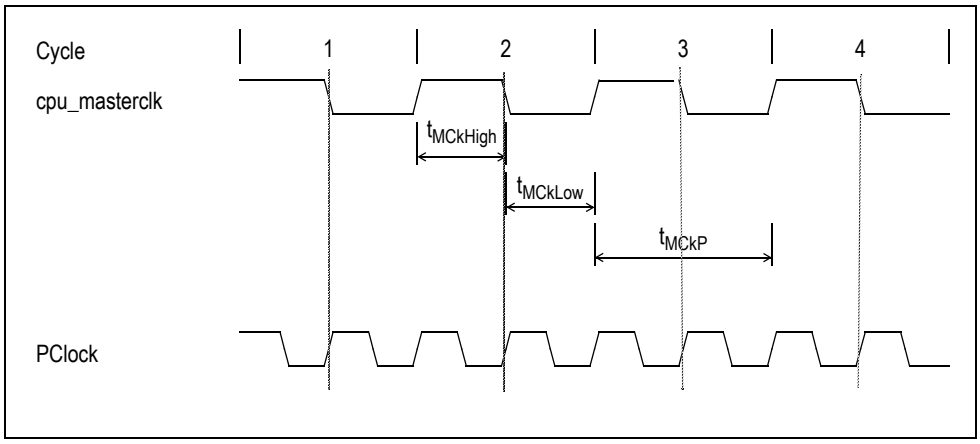


Figure 19.4 Timing Illustration of cpu_masterclk-to-PClock Multiply by 2

Phase-Locked Loop (PLL) Operation

The processor aligns the pipeline clock, PClock, to the cpu_masterclk by using an internal phase-locked loop (PLL) circuit that generates aligned clocks. By their nature, PLL circuits are only capable of generating aligned clocks for cpu_masterclk frequencies within a limited range.

Clocks generated using PLL circuits contain some inherent inaccuracy, or jitter; a clock aligned with cpu_masterclk by the PLL can lead or trail cpu_masterclk by as much as the maximum clock jitter specified in the clock parameters table in the data sheet for this device.

PLL Components and Operation

The storage capacitor required for the Phase-Locked Loop circuit is contained in the RC32334. However, it is recommended that the system designer provide a filter network of passive components for the PLL power supply.

The Phase Locked Loop circuit requires several passive components for proper operation, which are connected to Vcc, Vss, VccP, and VssP, as illustrated in Figure 19.5.

Notes

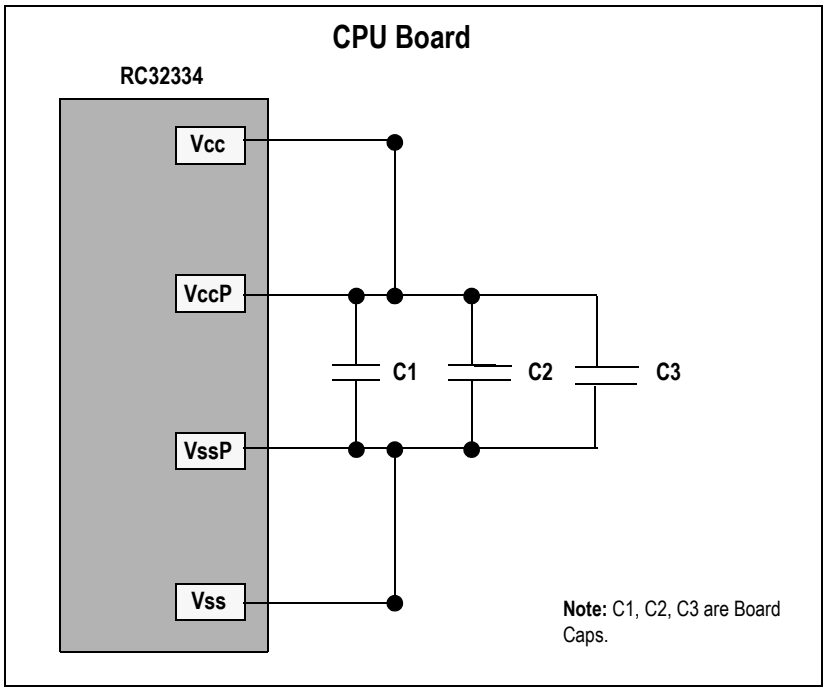


Figure 19.5 PLL Passive Components

It is essential to isolate the analog power and ground for the PLL circuit (**VccP**/**VssP**) from the regular power and ground (**Vcc**/**Vss**). Initial evaluations have yielded good results with the following values:

C1 = 1 nF

C2 = 3.3 µF

C3 = 10 µF

Because the optimum values for the filter components depend upon the application and the system noise environment, these values should be considered as starting points for further experimentation within your specific application.

PLL Analog Power Filtering

For noisy module environments, a filter circuit of the following form is recommended as shown in Figure 19.6.

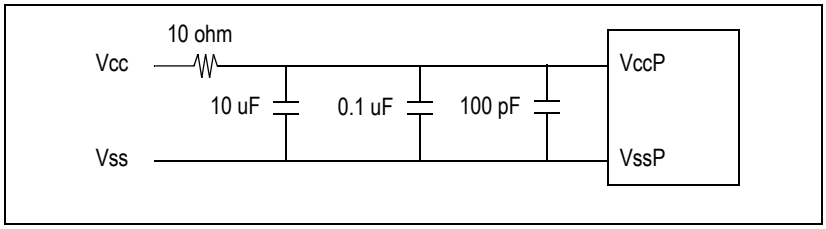


Figure 19.6 PLL Filter Circuit for Noisy Environments

Reset Function

The RC32334 reset uses the cpu_coldreset_n input signal:

- ***Power-on reset*** *starts when the power supply is turned on and completely re-initializes the internal state machine of the processor without saving any state information. Then, the ModeBit[9:0]are read, and the processor allows its internal phase locked loops to lock, stabilizing the processor internal clock. After the internal clock is stabilized, the reset exception will be taken. The timing of the cold reset signal is illustrated in Figure 19.7.*

Notes

Reset and Initialization Interface

During the reset sequence, the CPU RC32300 core of the RC32334 obtains configuration information using its mode configuration interface. The initialization values for the RC32334 are obtained from ejtag_pcst [2:0], mem_addr [19:17], debug_cpu_i_d_n, debug_cpu_ads_n, debug_cpu_dma_n and debug_cpu_ack_n signals which are ModeBit[9:0] during the power-on reset. The ModeBit[9:0] are latched with the rising edge (negating edge) of the cpu_coldreset_n signal. Timing of the mode configuration interface reset sequence is shown in Figure 19.7. Additional system controller configuration information is obtained from mem_addr[22:20] as explained in section Reset of On-chip System Controller Logic later in this chapter.

The boot-mode configuration settings are listed in Table 19.1.

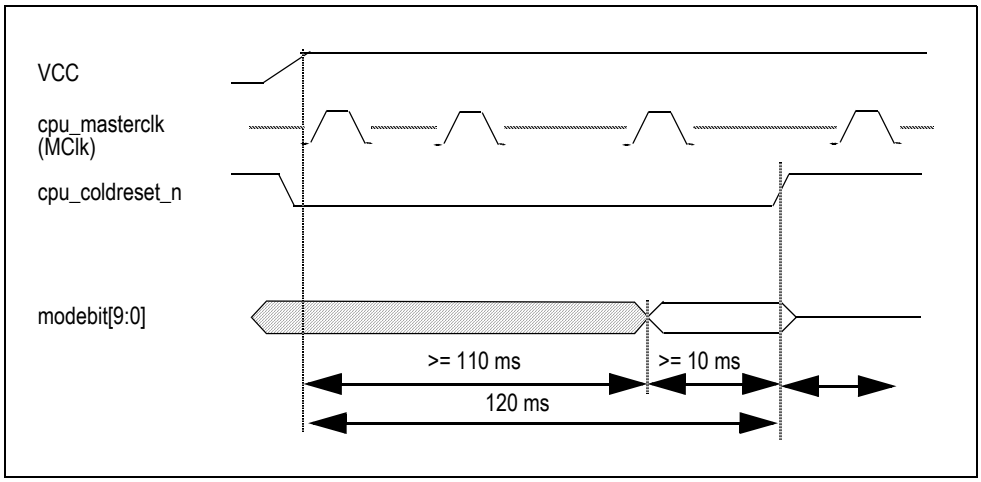


Figure 19.7 Mode Configuration Interface Reset Sequence

Boot-Mode Configuration Settings

Pin	Mode Bit	Description	Value	Mode Setting
ejtag_pcst[2:0]	2:0 MSB (2)	Clock Multiplier cpu_masterclk is multiplied internally to generate PClock	0	Multiply by 2
			1	Multiply by 3
			2	Multiply by 4
			3	Reserved
			4	Reserved
			5	Reserved
			6	Reserved
			7	Reserved
debug_cpu_i_d_n	3	EndBit	0	Little-endian ordering
			1	Big-endian ordering
debug_cpu_ack_n	4	Reserved	0	
debug_cpu_ads_n	5	Reserved	0	
debug_cpu_dma_n	6	TmrIntEn Enables/Disables the timer interrupt on cpu_int_n[5]	0	Enables timer interrupt
			1	Disables timer interrupt

Table 19.1 Boot-Mode Configuration Settings (Part 1 of 2)

Notes

Pin	Mode Bit	Description	Value	Mode Setting
mem_addr[17]	7	Reserved for future use.	1	
mem_addr[19:18]	9:8 MSB (9)	Boot-Prom Width Specifies the memory port width of the memory space which contains the boot prom.	00	8 bits
			01	16 bits
			10	32 bits
			11	Reserved

Table 19.1 Boot-Mode Configuration Settings (Part 2 of 2)

reset_boot_mode Settings

The RC32334 reset-boot mode initialization setting values and mode descriptions are listed in Table 19.2.

mem_addr[22:21]	mem_addr[20]	Description
1 1	x	Tri-state memory bus and EEPROM bus during coldreset_n assertion.
1 0	x	**Reserved**
0 1	1	**PCI-boot mode** (pci_host_mode must be in satellite mode) RC32334 will reset either from a cold reset or from a PCI reset. Boot code is provided via PCI. PCI is initialized via the PCI EEROM.
0 1	0	Not allowed
0 0	1	**Standard-boot mode (satellite mode)** Boot from the RC32334's memory controller (typical system). Serial EEPROM not supported.
0 0	0	**Standard-boot mode (host mode)** Boot from the RC32334's memory controller (typical system). Serial EEPROM not supported.

Table 19.2 RC32334 reset_boot_mode Initialization Settings

pci_host_mode Settings

During reset initialization, the RC32334's PCI interface can be set to the Satellite or Host mode settings. When set to the Host mode, the CPU must configure the RC32334's PCI configuration registers, including the read-only registers. If the RC32334's PCI is in the PCI-boot mode Satellite mode, read-only configuration registers are loaded by the serial EEPROM.

Reset of On-chip System Controller Logic

For the on-chip system logic, the reset sequence occurs in conjunction with the RC32300 CPU core reset sequence described above. On power up, cpu_coldreset_n is asserted. When cpu_coldreset_n is de-asserted, the RC32300 CPU core reset vector mode bits are latched in. This is followed by the reset vector from the on-chip system peripherals being latched in. After cpu_coldreset_n de-asserts, the cpu_reset_n signal is provided, back to the RC32300 CPU core and remains asserted for 256 clocks.

After cpu_reset_n de-asserts, the CPU will first issue the reset boot address at physical address 0x1fc0_0000, then initiate a boot memory cycle using mem_cs_n[0] set to 32 wait-states. The RC32334 uses mem_addr[22:17] to read in part of the reset vector. As shown in Figure 19.8, mem_addr[22:17] is tri-stated during coldreset and continues to be tri-stated until the 2nd clock after cpu_reset_n de-asserts. Typically, the system uses pull-up or pull-down resistors of 5K ohm to select the reset initialization vector when cpu_coldreset_n de-asserts.

Notes

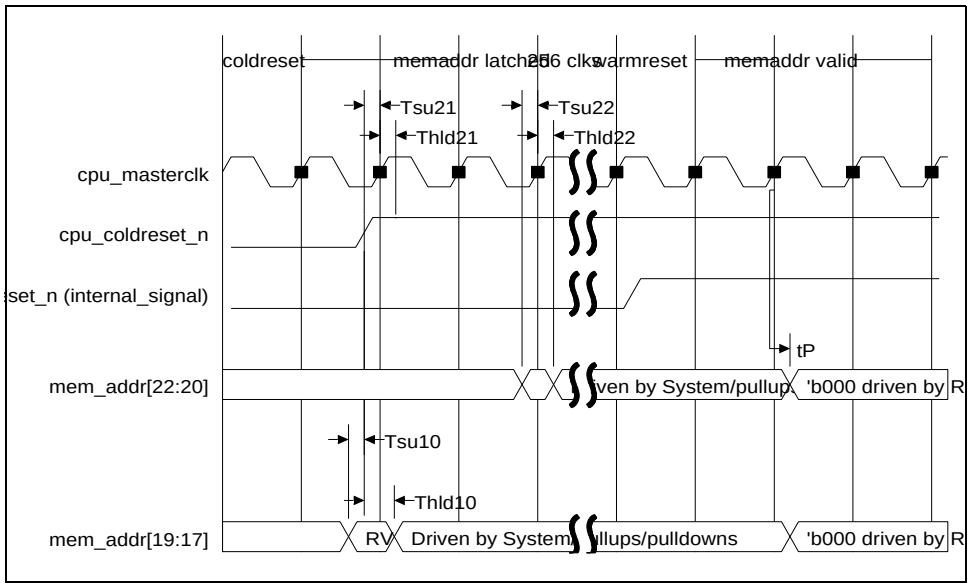


Figure 19.8 Reset Vector Initialization Part 1 of 2

The RC32334 usually drives debug_cpu_i_d_n, debug_cpu_ads_n, debug_cpu_dma_n, and debug_cpu_ack_n. During boot time, these four debug signals are used as reset initialization vector bits. Thus, the RC32334 tri-states these four debug signals during the assertion of cpu_coldreset_n, as shown in Figure 19.9. During coldreset, the system can pull-up or pull-down these four debug signals.

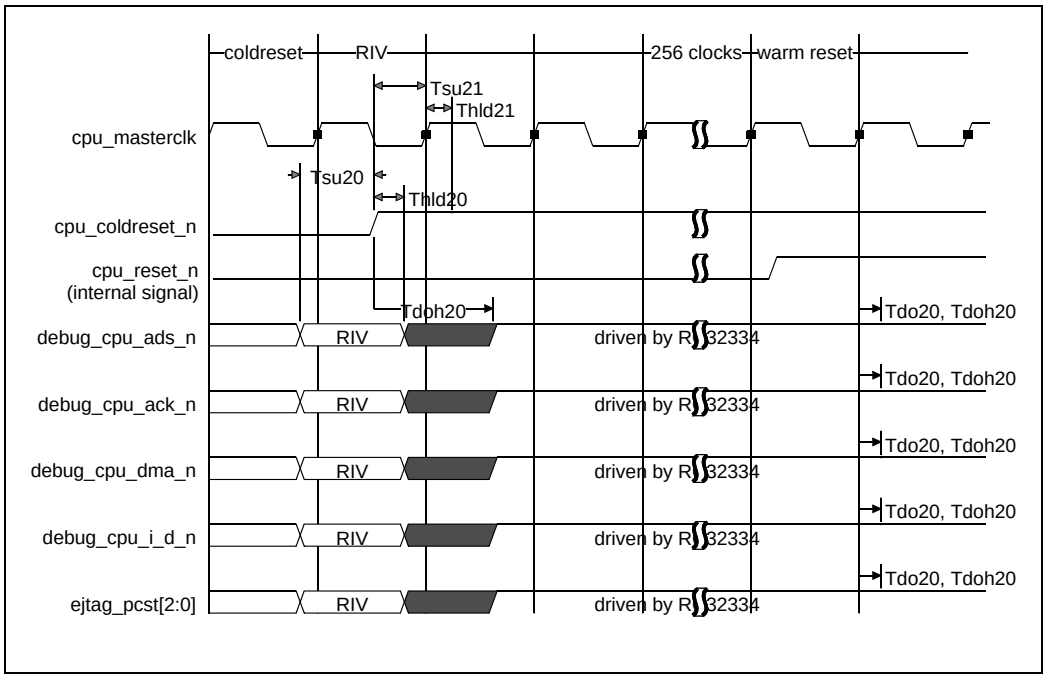


Figure 19.9 Reset Vector Initialization Part 2 of 2

JTAG Boundary Scan

Notes

Introduction

As previously described, the RC32334 is a logical integration of both the RC32364 standalone CPU and the RC32134 system controller. Because each of these discrete devices includes a TAP controller, there are 2 TAP controllers on the RC32334, one for the CPU core (referred to as the RC32300 CPU core TAP Controller), described in the next chapter, and one for System Logic controller, described in this chapter.

The System Controller TAP Controller is used to provide conventional standard JTAG Boundary Scan access to the RC32334 pin interface. The RC32300 CPU Core TAP Controller is used to provide access to the EJTAG interface on the CPU Core.

The two TAP Controllers are connected in parallel as shown in Figure 20.1 and share the JTAG control pins, except for separate jtag_tms and ejtag_tms pins. Thus at least one of the two TAP Controllers must be in Test-Logic-Reset at any given time, so that the jtag_tdo pin is only actively being driven from no more than one of the TAP Controllers. For example, if neither TAP Controller is in use, they both can be reset by asserting jtag_trst_n, or by asserting both jtag_tms and ejtag_tms high for 5 consecutive jtag_tck clocks. If the RC32300 CPU Core TAP Controller is to be used, then the System Controller TAP Controller must be reset by asserting jtag_tms high for 5 consecutive jtag_tck clocks. If the System Controller TAP Controller is to be used, then the RC32300 CPU Core TAP Controller must be reset by asserting ejtag_tms high for 5 consecutive jtag_tck clocks.

The RC32300 CPU Core TAP Controller is one of the two TAP Controllers on the RC32334. As such, the RC32300 CPU Core TAP Controller is used primarily for EJTAG support, since many EJTAG functions are accessed via the RC32300 CPU Core TAP Controller JTAG port. **Note that the Boundary Scan Register for the internal CPU Core must never be used, as it will access internally connected CPU Core ports/pins.** Instead the System Controller TAP Controller Boundary Scan Register is provided for RC32334 conventional JTAG pin access, control, and boundary scan.

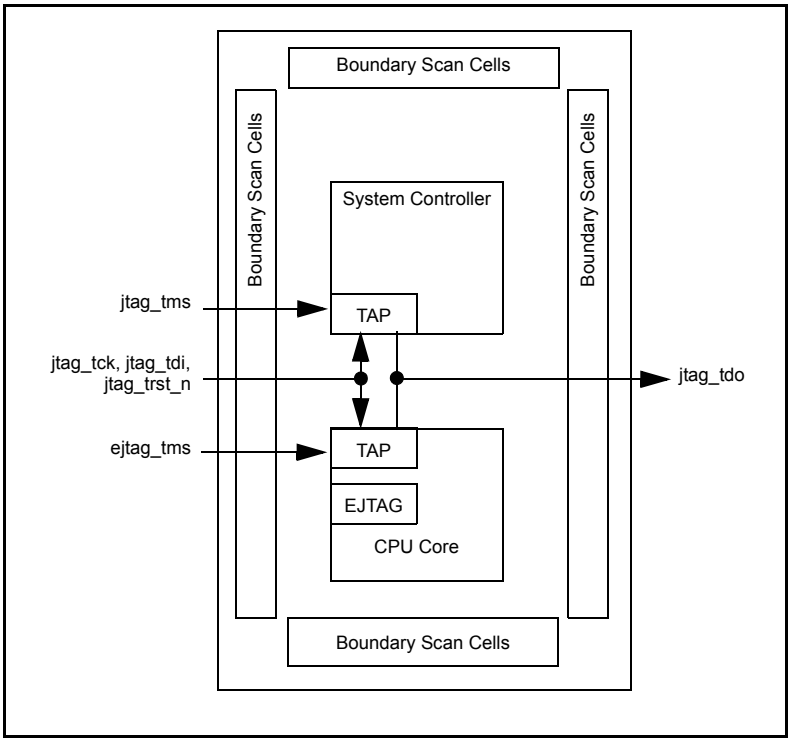


Figure 20.1 Dual TAP Controller Block Diagram

Notes

System Logic TAP Controller Overview

The system logic utilizes a 16-state, four-bit TAP controller, a four-bit instruction register, and five dedicated pins to perform a variety of functions. The primary use of the JTAG TAP Controller state machine is to allow the five external JTAG control pins to control and access the RC32334's many external signal pins. The JTAG TAP Controller can also be used for identifying the device part number. The JTAG logic of the RC32334 is depicted in Figure 20.2.

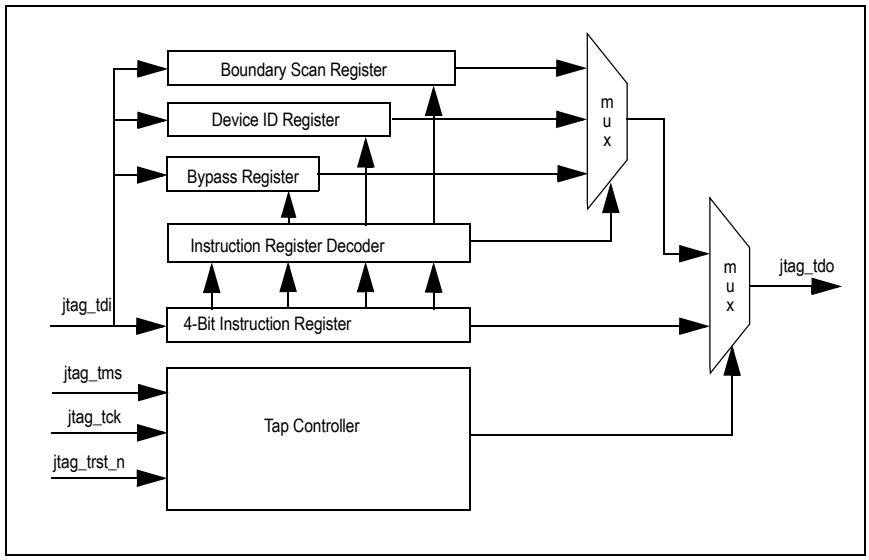


Figure 20.2 Diagram of the JTAG Logic

Signal Definitions

JTAG operations such as Reset, State-transition control and Clock sampling are handled through the signals listed in Table 20.1. A functional overview on the TAP Controller and Boundary Scan registers is provided in the sections following the table.

Pin Name	Type	Description
jtag_trst_n	Input	JTAG RESET Asynchronous reset for JTAG TAP.
jtag_tck	Input	JTAG Clock Test logic clock. Jtag_tms and jtag_tdi are sampled on the rising edge. Jtag_tdo is output on the falling edge.
jtag_tms	Input	JTAG Mode Select Requires an external pull-up. Controls the state transitions for the TAP controller state machine (internal pull-up).
jtag_tdi	Input	JTAG Input Serial data input for BSC chain, Instruction Register, IDCODE register, and BYPASS register (internal pull-up).
jtag_tdo	Output	JTAG Output Serial data out. Tri-stated except when shifting while in Shift-DR and SHIFT-IR TAP controller states.

Table 20.1 JTAG Pin Descriptions

The system logic TAP controller transitions from state to state, according to the value present on jtag_tms, as sampled on the rising edge of jtag_tck. The Test-Logic Reset state can be reached either by asserting jtag_trst_n or by applying a 1 to jtag_tms for five consecutive cycles of jtag_tck. A state diagram for the TAP controller appears in Figure 20.3. The value next to state represent the value that must be applied to jtag_tms on the next rising edge of jtag_tck, to transition in the direction of the associated arrow.

Notes

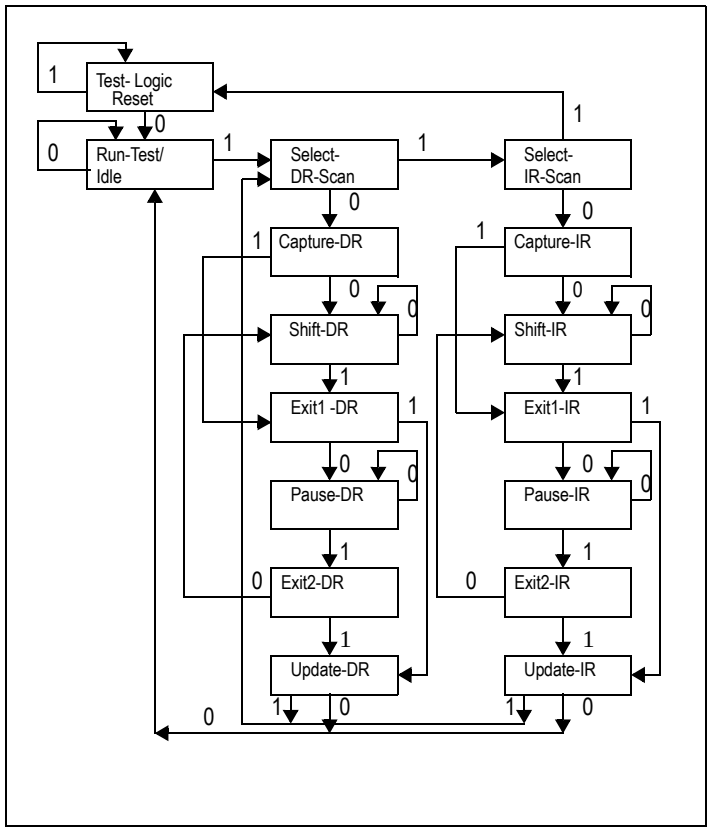


Figure 20.3 State Diagram of RC32334's TAP Controller

Test Data Register (DR)

The Test Data register contains the following:

- *The Bypass register*
- *The Boundary Scan registers*
- *The Device ID register*

These registers are connected in parallel between a common serial input and a common serial data output, and are described in the following sections. For more detailed descriptions, refer to IEEE Standard Test Access port (IEEE Std. 1149.1-1990).

Boundary Scan Registers

The RC32334 scan chain is 330 bits long and comprises 171 logical elements--where each logical element represents a signal pin. The five JTAG pins do not have scan elements associated with them, nor does the EJTAG ejtag_tms pin. Of the 171 logical elements, 125 are two-bit bidirectional cells, 33 are two-bit tri-statable outputs, and 14 are one-bit dedicated inputs.

The RC32332 scan chain is 303 bits long, has 157 elements: 116 bidirectional, 30 outputs, 11 inputs. This boundary scan chain is connected between jtag_tdi and jtag_tdo when the EXTEST or SAMPLE/PRELOAD instructions are selected. Once EXTEST is selected and the TAP controller passes through the UPDATE-IR state, whatever value is currently held in the boundary scan register's output latches is immediately transferred to the corresponding outputs or output enables.

Therefore, the SAMPLE/PRELOAD instruction must first be used to load suitable values into the boundary scan cells, so that inappropriate values are not driven out onto the system pins. All of the boundary scan cells feature a negative edge latch, which guarantees that clock skew cannot cause incorrect data to be latched into a cell. The input cells are sample-only cells. The simplified logic configuration is shown in Figure 20.4.

Notes

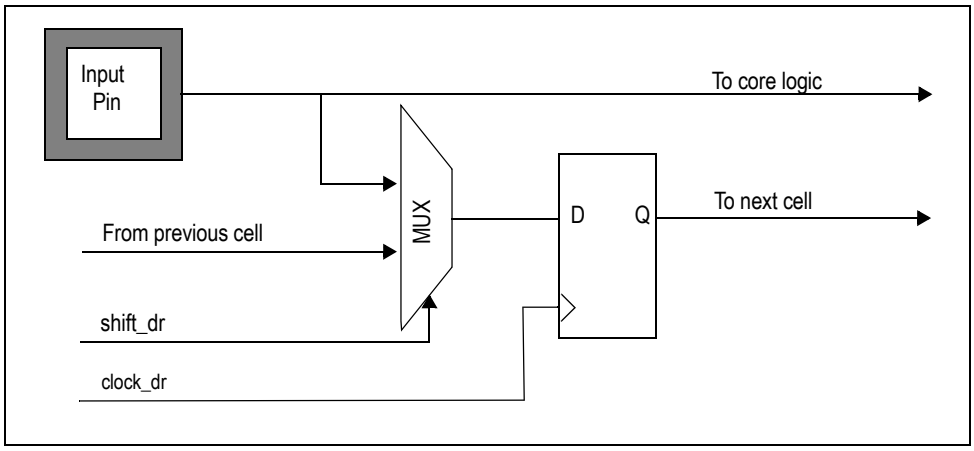


Figure 20.4 Diagram of Observe-only Input Cell

The simplified logic configuration of the output cells is shown in Figure 20.5.

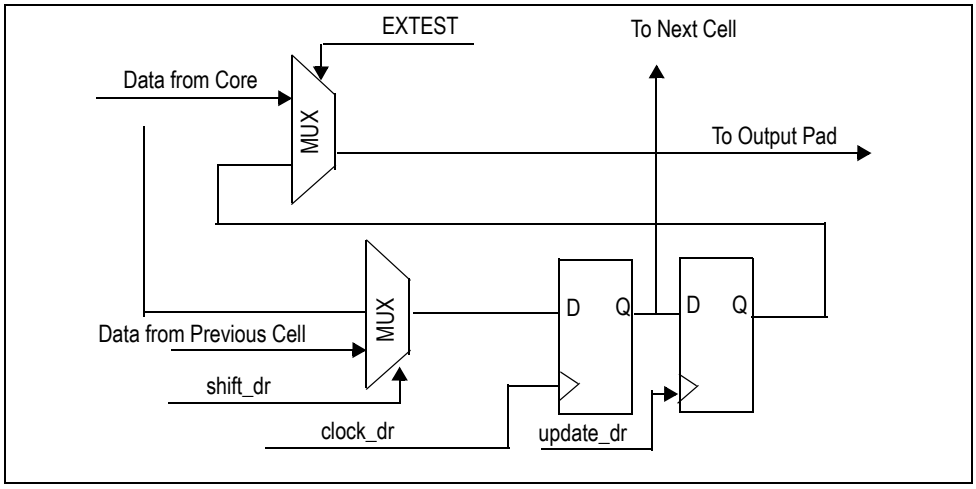


Figure 20.5 Diagram of Output Cell

The output enable cells are also basically output cells. The simplified logic appears in Figure 20.6.

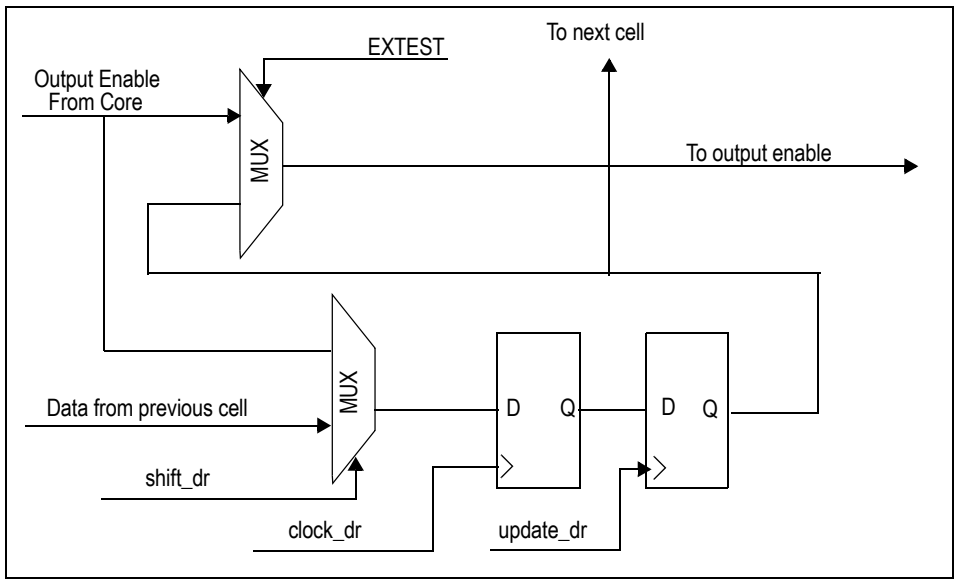


Figure 20.6 Diagram of Output Enable Cell

Notes

The bidirectional cells are composed of only two boundary scan cells. They contain one output enable cell and one capture cell, which contains only one register. The input to this single register is selected via a mux that is driven selected by the output enable cell and the EXTEST mode. When the output enable cell is driving a high out to the pad (which enables the pad for output) and EXTEST is disabled, the single capture register will be configured to capture from the output signal from the core to the pad.

However, in the case where the Output Enable is low, signifying a tri-state condition at the pad, then the Capture Register will capture from the input from the pad. The configuration is shown graphically in Figure 20.7.

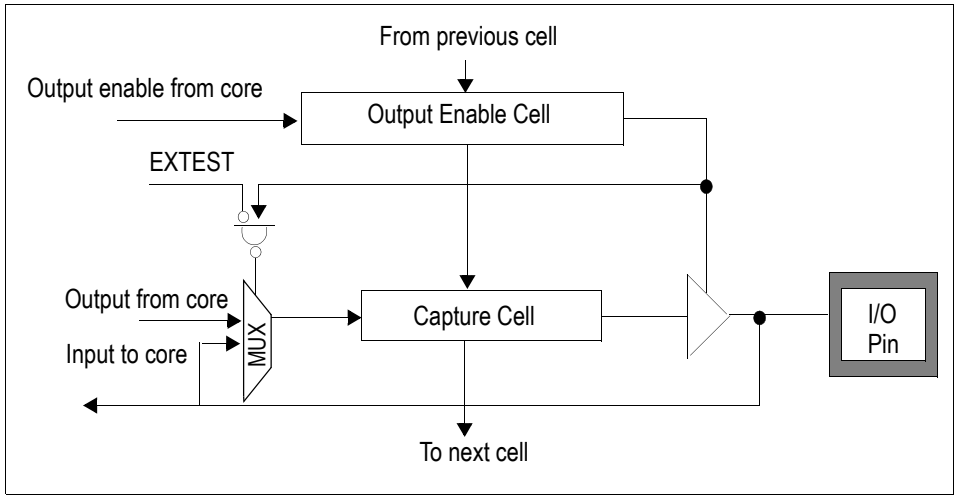


Figure 20.7 Diagram of Bidirectional Cell

Instruction Register (IR)

The Instruction register allows an instruction to be shifted serially into the processor at the rising edge of jtag_tck. The instruction is then used to select the test to be performed or the test register to be accessed, or both. The instruction shifted into the register is latched at the completion of the shifting process, when the TAP controller is at the Update-IR state.

The Instruction register contains four shift-register-based cells that can hold instruction data. These mandatory cells are located near the serial outputs and are the least significant bits. The values of the bits are 0 and 1 (1 is the least significant bit). This register is decoded to perform the following functions:

- *To select test data registers that may operate while the instruction is current. The other test data registers should not interfere with chip operation and selected data registers.*
- *To define the serial test data register path used to shift data between jtag_tdi and jtag_tdo during data register scanning.*

The Instruction Register is comprised of 4 bits to decode instructions as follows in Table 20.2.

Instruction	Definition	Opcode
EXTEST	Mandatory instruction allowing the testing of board level interconnections. Data is typically loaded onto the latched parallel outputs of the boundary scan shift register using the SAMPLE/PRELOAD instruction prior to use of the EXTEST instruction. EXTEST will then hold these values on the outputs while being executed. Also see the CLAMP instruction for similar capability.	0000

Table 20.2 Instructions Supported By RC32334's JTAG Boundary Scan (Part 1 of 2)

Notes

Instruction	Definition	Opcode
SAMPLE/ PRELOAD	Mandatory instruction that allows data values to be loaded onto the latched parallel output of the boundary-scan shift register prior to selection of the other boundary-scan test instruction. The Sample instruction allows a snapshot of data flowing from the system pins to the on-chip logic or vice versa.	0001
DEVICE_ID	Provided to select Device Identification to read out manufacturers identity, part, and version number	0010
HIGHZ	Tri-states all output and bidirectional boundary scan cells.	0011
RESERVED	Behaviorally equivalent to the BYPASS instruction as per the IEEE std. 1149.1 specification. However, the user is advised to use the explicit BYPASS instruction.	0100
RESERVED		0101
RESERVED		0110
RESERVED		0111
CLAMP	Provides JTAG user the option to bypass the part's JTAG controller while keeping the part outputs controlled similar to EXTEST.	1000
UNUSED	The unused instructions are behaviorally equivalent to the BYPASS instruction as per the IEEE Std. 1149.1 specification. However, the user is advised to use the explicit BYPASS instruction, as the internal usage of these currently unused instructions could possibly vary in future implementations of the device.	1001
UNUSED		1010
UNUSED		1011
UNUSED		1100
VALIDATE	Automatically loaded into the instruction register whenever the TAP controller passes through the CAPTURE-IR state. The lower two bits '01' are mandated by the IEEE std. 1149.1 specification.	1101
UNUSED	Same as other UNUSED instructions above	1110
BYPASS	The BYPASS instruction is used to truncate the boundary scan register as a single bit in length	1111

Table 20.2 Instructions Supported By RC32334's JTAG Boundary Scan (Part 2 of 2)

Extest

The external test (EXTEST) instruction is used to control the boundary scan register, once it has been initialized using the SAMPLE/PRELOAD instruction. Using EXTEST, the user can then sample inputs from or load values onto the external pins of the RC32334. Once this instruction is selected, the user then uses the SHIFT-DR TAP controller state to shift values into the boundary scan chain. When the TAP controller passes through the UPDATE-DR state, these values will be latched onto the output pins or into the output enables.

Sample/Preload

The sample/preload instruction has a dual use. The primary use of this instruction is for preloading the boundary scan register prior to enabling the EXTEST instruction. Failure to preload will result in unknown random data being driven onto the output pins when EXTEST is selected. The secondary function of SAMPLE/PRELOAD is for sampling the system state at a particular moment. Using the SAMPLE function, the user can halt the device at a certain state and shift out the status of all of the pins and output enables at that time.

Bypass

The BYPASS instruction is used to truncate the boundary scan register to a single bit in length. During system level use of the JTAG, the boundary scan chains of all the devices on the board are connected in series. In order to facilitate rapid testing of a given device, all other devices are put into BYPASS mode.

Notes

Therefore, instead of having to shift 307 times to get a value through the RC32334, the user only needs to shift one time to get the value from jtag_tdi to jtag_tdo. When the TAP controller passes through the CAPTURE-DR state, the value in the BYPASS register is updated to be 0.

If the device being used does not have a DEVICE_ID register, then the BYPASS instruction will automatically be selected into the instruction register whenever the TAP controller is reset. Therefore, the first value that will be shifted out of a device without a DEVICE_ID register is always 0. Devices such as the RC32334 that include a DEVICE_ID register will automatically load the DEVICE_ID instruction when the TAP controller is reset, and they will shift out an initial value of 1. This is done to allow the user to easily distinguish between devices having DEVICE_ID registers and those that do not.

Clamp

This instruction, listed as optional in the IEEE 1149.1 JTAG Specifications, allows the boundary scan chain outputs to be clamped to fixed values. When the clamp instruction is issued, the scan chain will bypass the RC32334 and pass through to devices further down the scan chain.

DeviceID

The DEVICEID instruction is automatically loaded when the TAP controller state machine is reset either by the use of the jtag_trst_n signal or by the application of a '1' on jtag_tms for five or more cycles of jtag_tck as per the IEEE Std 1149.1 specification. The least significant bit of this value must always be 1. Therefore, if a device has a DEVICE_ID register, it will shift out a 1 on the first shift if it is brought directly to the SHIFT-DR TAP controller state after the TAP controller is reset. The board- level tester can then examine this bit and determine if the device contains a DEVICE_ID register (the first bit is a 1), or if the device only contains a BYPASS register (the first bit is 0).

However, even if the device contains a DEVICE_ID register, it must also contain a BYPASS register. The only difference is that the BYPASS register will not be the default register selected during the TAP controller reset. When the DEVICE_ID instruction is active and the TAP controller is in the Shift-DR state, the thirty-two bit value that will be shifted out of the device-ID register is `0x10018067`.

Bit(s)	Mnemonic	Description	R/W	Reset
0	reserved	reserved 0x1	R	1
11:1	Manuf_ID	**Manufacturer Identity** (11 bits) IDT 0x33	R	0x33
27:12	Part_number	**Part Number** (16 bits) This field identifies the part number of the processor derivative. For the RC32334 this value is: 0x0018 For the RC32332 this value is: 001Ah	R	impl. dep.
31:28	Version	**Version** (4 bits) This field identifies the version number of the processor derivative. For the RC32334/RC32332, this value is 0x1	R	impl. dep.

Table 20.3 System Controller Device Identification Register

Version	Part Number	Vendor ID	LSB
0001	0000|0000|0001|1000	0000|0110|011	1

Figure 20.8 System Controller Device ID Instruction Format

Validate

The VALIDATE instruction is automatically loaded into the instruction register whenever the TAP controller passes through the CAPTURE-IR state. The lower two bits '01' are mandated by the IEEE Std. 1149.1 specification.

Notes

Reserved

Reserved instructions implement various test modes used in the device manufacturing process. The user should not enable these instructions.

Unused[1]

The unused instructions are behaviorally equivalent to the BYPASS instruction as per the IEEE Std. 1149.1 specification. However, the user is advised to use the explicit BYPASS instruction as the internal usage of these currently unused instructions could possibly vary in future implementations of the device.

Usage Considerations

As previously stated, there are internal pull-ups on jtag_trst_n, jtag_tms, and jtag_tdi. However, jtag_tck also needs to be driven to a known value. However, it is best to drive a zero on the jtag_tck pin when it is not in use or use an external pull-down resistor. In order to guarantee that the JTAG does not interfere with normal system operation, the TAP controller should be forced into the Test-Logic-Reset controller state by continuously holding jtag_trst_n low and/or jtag_tms high when the chip is in normal operation. If JTAG will not be used, externally pull-down jtag_trst_n low to disable it.

1. Any unused instruction is defaulted to the BYPASS instruction

EJTAG (In-circuit Emulator) Interface

Notes

Introduction

As previously described, the RC32334 is a logical integration of both the RC32364 stand-alone CPU and the RC32134 system controller. Because each of these discrete devices includes a TAP controller, there are 2 TAP controllers on the RC32334, one for the CPU core (referred to as the RC32300 CPU core TAP Controller), described in this chapter, and one for System Logic Controller, described in the previous chapter.

The System Logic Controller TAP Controller is used to provide conventional standard JTAG Boundary Scan access to the RC32334 pin interface. The RC32300 CPU core TAP Controller is used to provide access to the EJTAG interface on the CPU core. The EJTAG version implemented in the RC32334 is 1.5.3.

The two TAP Controllers are connected in parallel as shown in Figure 21.1 and share the JTAG control pins, except for separate jtag_tms and ejtag_tms pins. Thus at least one of the two TAP Controllers must be in Test-Logic-Reset at any given time, so that the jtag_tdo pin is only actively being driven from no more than one of the TAP Controllers. Thus for example, if neither TAP Controller is in use, they both can be reset by asserting jtag_trst_n, or by asserting both jtag_tms and ejtag_tms high for 5 consecutive jtag_tck clocks. If the RC32300 CPU core TAP Controller is to be used, then the System Controller TAP Controller must be reset by asserting jtag_tms high for 5 consecutive jtag_tck clocks. If the System Controller TAP Controller is to be used, then the RC32300 CPU core TAP Controller must be reset by asserting ejtag_tms high for 5 consecutive jtag_tck clocks.

Note that the Boundary Scan Register for the internal CPU Core must never be used, as it will access internally connected CPU Core ports/pins. Instead the System Logic Controller TAP Controller Boundary Scan Register is provided for RC32334 conventional JTAG pin access, control, and boundary scan.

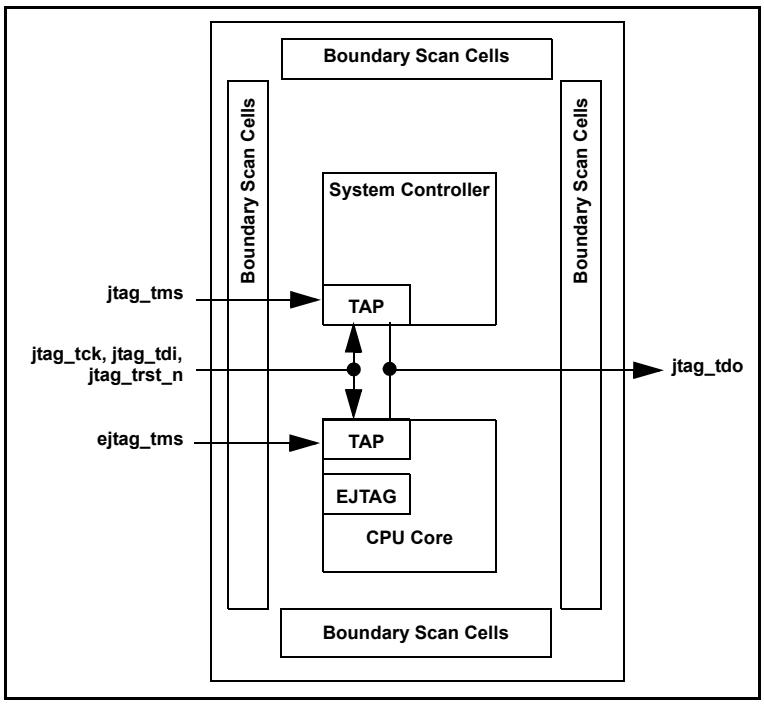


Figure 21.1 Dual TAP Controller Block Diagram

Notes

On-chip support for low-cost in-circuit emulation (ICE) equipment is featured on the RC32334. The RC32300 CPU core on the RC32334 implements the standard MIPS Enhanced JTAG (EJTAG) interface, which includes the following key ICE interface capabilities:

- *Breakpoints*
- *Debug exception handlers*
- *Execution trace capability*

Overview

The following features are supported by the EJTAG:

- *Two additional instructions are added to the RC32300 CPU core: Set Software Debug Breakpoints (SDBBP) and Return from Debug Exception (DERET).*
- *The EJTAG module doesn't support single step execution in hardware. However, it can be accomplished in software.*
- *Hardware breakpoints can be set at:*
  - *virtual instruction address (with address bit masking)*
  - *virtual data address (with address bit masking) and data value (with byte lane masking)*
  - *physical processor core address (with lower address bit masking) and physical processor core data (with data bit masking)*
- *Trace Trigger points can be specified instead of hardware breakpoints. The trace trigger is limited by the max speed of the ejtag_dclk that the EJTAG probe can sustain.*
- *Debug breaks can be initiated by the EJTAG Probe via a JTAG pin (*jtag_tdi / ejtag_dint_n*).*
- *PC Trace information is provided by additional status pins and the processor clock.*

The EJTAG unit on the RC32300 CPU core is used for debugging the state of the CPU core and is unaware of the peripherals around the core (memory controller, DRAM controller, etc.) that are used to create the RC32334. To access the peripherals around the CPU core, the ICE probe must execute standard load and store instructions to interrogate the register contents of these modules.

The block diagram of the EJTAG Unit on the RC32300 CPU is given in Figure 21.2, and the simplified block diagram is shown in Figure 21.3.

The following main blocks provide debug functionality:

- *Instruction Address Match Logic*
- *Data Address & Data Value Match Logic*
- *Processor Address Bus & Processor Data Bus Match Logic*
- *PC Trace Logic*
- *Software Debug Breakpoint (SDBBP) instruction and Debug Exception Return (DERET) instruction*
- *Debug Registers*

Notes

Block Diagrams

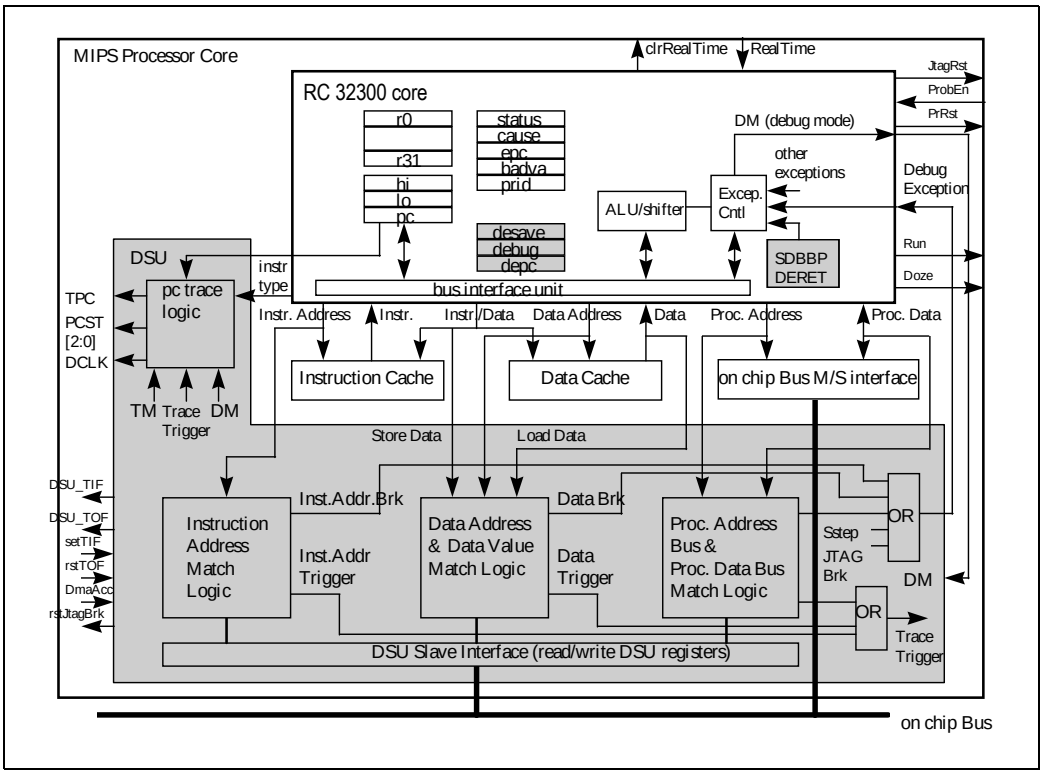


Figure 21.2 Block Diagram

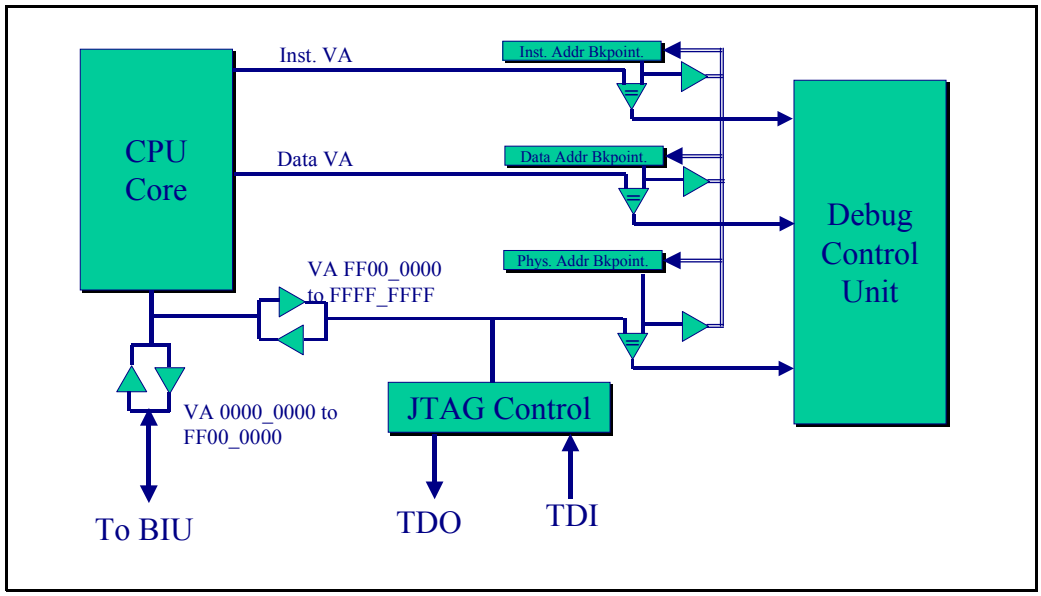


Figure 21.3 Simplified EJTAG Block Diagram

Debug Support Unit

This section describes the EJTAG Debug Support unit. It covers the debug instructions added to the RC32300 CPU core instruction set as well as support functions and registers for debugging.

The debug unit is used to access the internal state of the RC32300 CPU Core, through a standard JTAG interface that is compatible with the IEEE Std. 1149.1 specification (refer to Chapter 20 in this manual for additional information). Additional status pins (for Run-time and Real-time data collection) along with

Notes

external third-party hardware and software creates an enhanced JTAG interface - referred to as EJTAG - which provides a Real-Time debugging system.

Information on instructions that have been added to the MIPS ISA instruction set and Real-time debugging register descriptions are also included.

Instruction Address Match Logic

If a match occurs between the processor's virtual Instruction Address and the address value set in the Instruction Address Break register, a Debug Exception is generated to the core and/or a Trace Trigger code is applied to the ejtag_pcst[2:0] lines. Address bits can be excluded from comparison by setting mask bits in a Mask register.

Data Address & Data Value Match Logic

If a match occurs between the processor's virtual Data Address and the address value set in the Data Address Break register, then a Debug Exception is generated to the core and/or a Trace Trigger code is applied to the ejtag_pcst[2:0] lines. Status bits in the Debug register indicate load or store access. Address bits can be excluded from comparison by setting mask bits in a Mask register.

Processor Address Bus & Processor Data Bus Match Logic

If a match occurs between the Processor's physical Address Bus and the address value set in the Processor Address Bus Break register *and* there is also a match between the processor's accompanying data and the value in the Processor Data Bus Break register, then a Debug Exception is generated to the core and/or a Trace Trigger code is applied to the ejtag_pcst[2:0] lines. The lower 24 Address bits can be excluded from comparison by setting mask bits in a Mask register; the Processor Data Bus bits can be excluded from comparison by setting mask bits in a Mask register.

The hardware Match Logic is not the only way to generate a Debug Exception. It can also be accomplished by the SDBBP instruction and by the EJTAG Probe (through JTAG).

The cause of the Debug Exception can be found in status bits of the Debug Register.

EJTAG Interface

The EJTAG interface consists of the standard JTAG signals (i.e. jtag_tck, jtag_tms, jtag_tdi, jtag_tdo, jtag-trst), extended with extra signals that provide real time program counter output. A description of the EJTAG pins is shown in Table 21.1.

Name	Type	Drive Strength/ Capability	Description
jtag_tck	Input	—	**JTAG Test Clock** Requires an external pull-down. An input test clock used to shift into or out of the Boundary-Scan register cells. jtag_tck is independent of the system and the processor clock with nominal 50% duty cycle.
jtag_tdi, ejtag_dint_n	Input	—	**JTAG Test Data In** Requires an external pull-up. On the rising edge of jtag_tck, serial input data are shifted into either the Instruction or Data register, depending on the TAP controller state. During Real Mode, this input is used as an interrupt line to stop the debug unit from Real Time mode and return the debug unit back to Run Time Mode (standard JTAG). This pin is also used as the ejtag_dint_n signal in the EJTAG mode.

Table 21.1 EJTAG Pins (Part 1 of 2)

Notes

Name	Type	Drive Strength/ Capability	Description
jtag_tdo, ejtag_tpc	Output	High	**JTAG Test Data Out** The jtag_tdo is serial data shifted out from instruction or data register on the falling edge of jtag_tck. When no data is shifted out, the jtag_tdo is tri-stated. During Real Time Mode, this signal provides a non-sequential program counter at the processor clock or at a division of processor clock. This pin is also used as the ejtag_tpc signal in the EJTAG mode.
jtag_tms	Input	—	**JTAG Test Mode Select** Requires an external pull-up. The logic signal received at the jtag_tms input is decoded by the TAP controller to control test operation. jtag_tms is sampled on the rising edge of the jtag_tck.
jtag_trst_n	Input	—	**JTAG Test Reset** The jtag_trst_n pin is an active-low signal for asynchronous reset of the debug unit, independent of the processor logic. An external pull-up on the board is recommended to meet the JTAG specification in cases where the tester can not access this signal. However, specific systems ordinarily should either: 1) drive low this signal 2) use an external pull-down on the board 3) clock jtag_tclk
ejtag_dclk	Output	—	**EJTAG Test Clock** Processor Clock. During Real Time Mode, this signal is used to capture address and data from the ejtag_tpc signal at the processor clock speed or any division of the internal pipeline.
ejtag_pcst[2:0]	I/O	Low	**EJTAG PC Trace Status Information** 111 (STL) Pipe line Stall 110 (JMP) Branch/Jump forms with PC output 101 (BRT) Branch/Jump forms with no PC output 100 (EXP) Exception generated with an exception vector code output 011 (SEQ) Sequential performance 010 (TST) Trace is outputted at pipeline stall time 001 (TSQ) Trace trigger output at performance time 000 (DBM) Run Debug Mode Alternate function: modebit[2:0].
ejtag_debugboot	Input	—	**EJTAG DebugBoot** Requires an external pull-down. The ejtag_debugboot input is used during reset and forces the CPU core to take a debug exception at the end of the reset sequence instead of a reset exception. This enables the CPU to boot from the ICE probe without having the external memory working. This input signal is level sensitive and is not latched internally. This signal will also set the JtagBrk bit in the JTAG_Control_Register[12].
ejtag_tms	Input	—	**EJTAG Test Mode Select** Requires an external pull-up. The ejtag_tms is sampled on the rising edge of jtag_tck.

Table 21.1 EJTAG Pins (Part 2 of 2)

Note: All input signals require pull-ups at the bonding pads per the JTAG specifications.

Note: The sharing of the JTAG pins for scan chain and debug requires that the scan chain of the board, if used, is disconnected from the EJTAG interface when it is being used for debugging.

Operating Modes

The RC32300 CPU core has two operating modes: **Normal mode** and **Debug mode**. The *Normal mode* is when the processor is *not* executing the debug exception handler routine.

Notes

Figure 21.4 shows a state diagram where the processor modes and the EJTAG interface functions are indicated.

The Debug mode is entered after a Debug Exception (derived from hardware breakpoints, single step etc.) is taken and continues until the Debug Exception Return (DERET) has been executed. In this time the processor is executing the Debug Exception handler routine.

In Debug Mode, the ProbEn bit determines the debug exception vector address: in case ProbEn=0 the debug exception handler starts at 0xBFC0-0480 and a debug monitor program will be entered and executed through regular system memory. When ProbEn=1, the debug exception vector address is 0xFF20-0200 and a debug monitor program can be executed in a “serial” way through the EJTAG protocol. (In this case, the monitor program is located on the EJTAG Probe, not requiring any physical EPROM on the target board).

In Debug Mode mode the standard IEEE 1149.1 Test Access Port (TAP) interface (referred to as JTAG) is used to control the on-chip debug support unit block (DSU). All operations such as read and write to internal registers, to external system memories and to other on-chip peripherals is performed by the EJTAG protocol. In this case, the pins jtag_tdi/ejtag_dint_n* and jtag_tdo/ejtag_tpc function as jtag_tdi input and jtag_tdo output.

By executing a PC Trace instruction, defined as an extended JTAG instruction, the PC Trace mode is entered. This can only be done in Debug Mode and when the EJTAG Probe is present (ProbEn=1). Prior to execution of the PC Trace instruction the TAP controller must be placed in Run-Test/Idle state by toggling the jtag_tms signal. In PC Trace mode, Program counter trace information is output via additional status pins in conjunction with the JTAG pins jtag_tdi/ejtag_dint_n* and jtag_tdo/ejtag_tpc. These pins now function as ejtag_dint_n* input and ejtag_tpc output. Non-sequential program counter data is available at the jtag_tdo/ejtag_tpc pin clocked out at the processor speed using the ejtag_dclk pin. The type of execution is available as status at the ejtag_pcst[2:0] pins. The PC Trace mode can be switched off by a Debug Exception caused e.g. by a breakpoint or when the EJTAG Probe activates the interrupt signal at the jtag_tdi/ejtag_dint_n* pin (which sets the JtagBrk bit in the EJTAG_Control_register[12]). When the PC Trace mode is switched off by a debug exception, the JTAG instruction register will be set to the BYPASS code (0x1F).

Notes

JTAG Operation

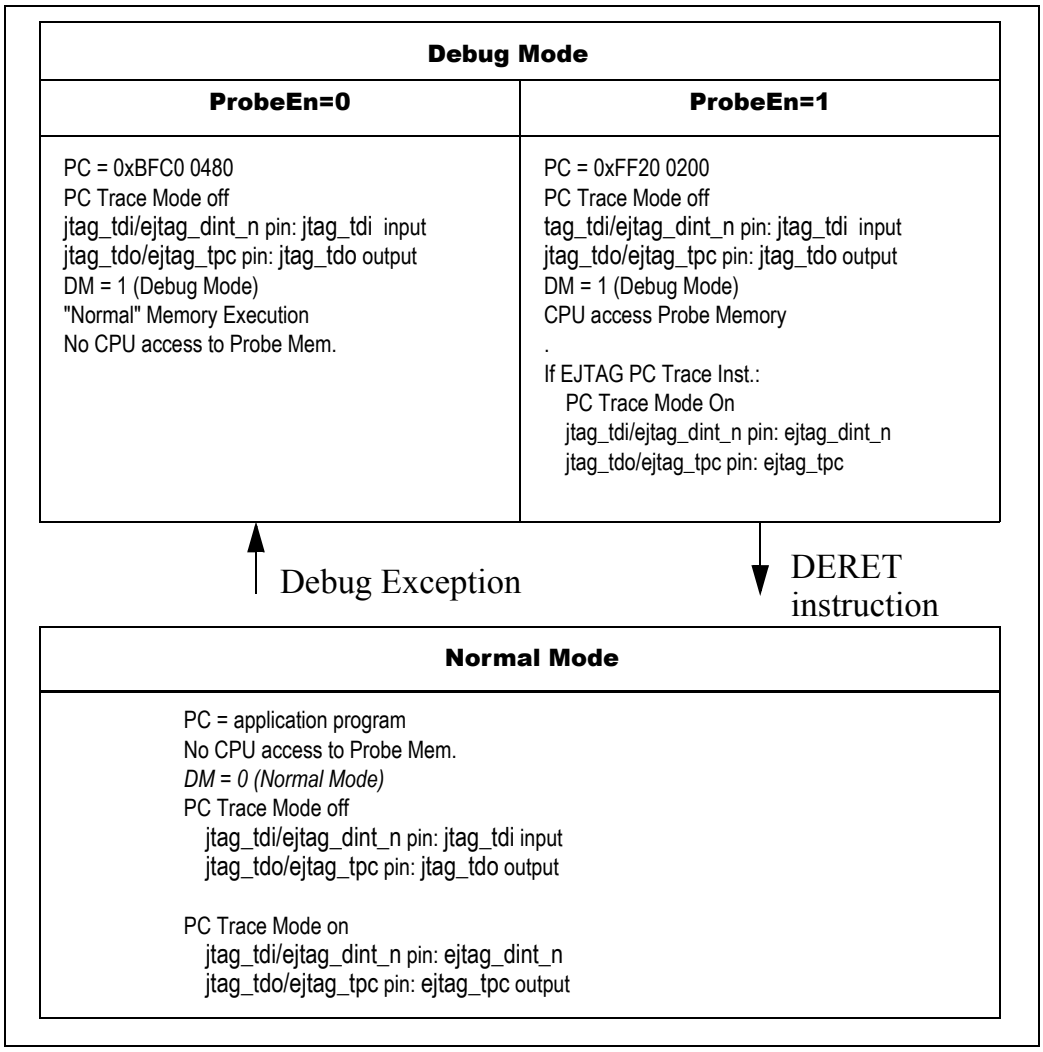


Figure 21.4 RC32334 Debug Operating Modes

Test Interface and Boundary-Scan Architecture

The IEEE 1149.1 architecture is shown in the shaded part of Figure 21.2. It consists of an Instruction Register, a Bypass Register, a Device ID register, an Implementation register and several User Data Registers (like the EJTAG Address/Data/Control registers) and a test interface referred to as a Test Access Port (TAP) controller.

The Instruction Register and Data Registers are separate scan paths arranged between the primary Test Data Input (jtag_tdi) pin and primary Test Data Output (jtag_tdo) pin. This architecture allows the TAP controller to select and shift data through one of the two types of scan paths, instruction or data, without accessing the other scan path.

Test Access Port Operation

The TAP controller is controlled by the Test Clock (jtag_tck) and Test Mode Select (ejtag_tms) inputs. These two inputs determine whether an Instruction Register scan or Data Register scan is performed. The TAP consists of a small controller design, driven by the jtag_tck input, which responds to the ejtag_tms input as shown in the state diagram in Figure 21.5. The IEEE 1149.1 test bus uses both clock edges of jtag_tck. jtag_tms and jtag_tdi are sampled on the rising edge of jtag_tck, while jtag_tdo changes on the fall

The state diagram for the TAP Controller is shown in Figure 21.5.

Notes

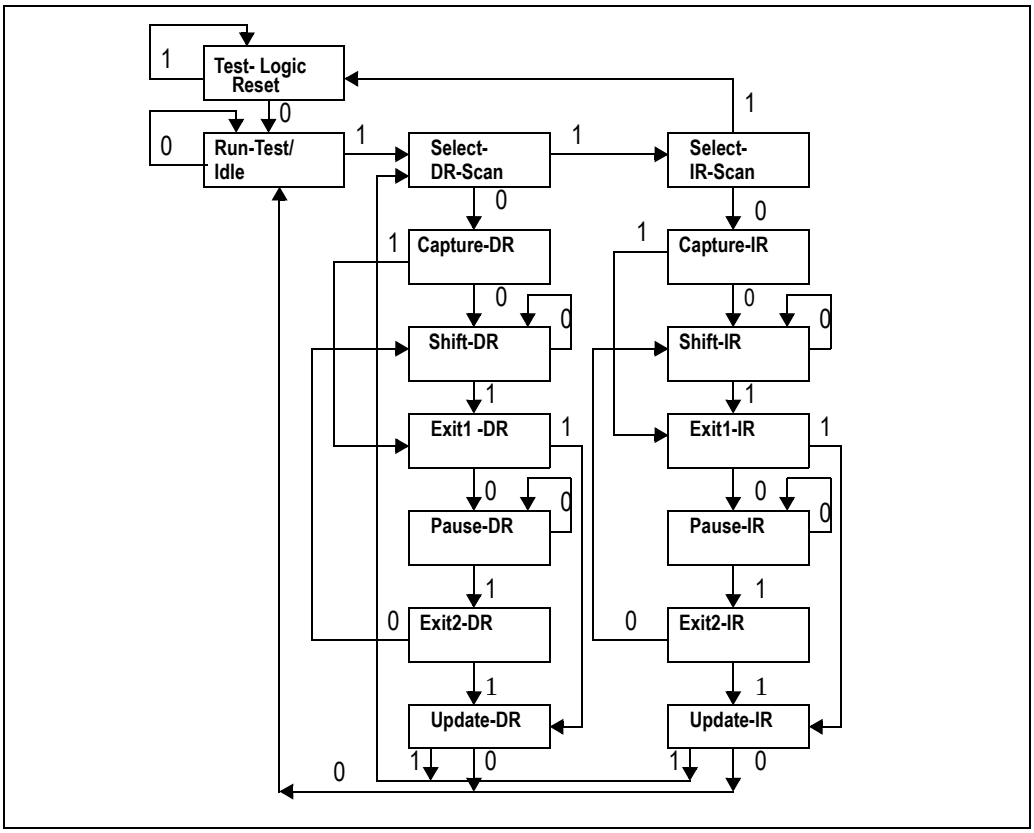


Figure 21.5 TAP Controller State Diagram

Refer to IEEE Standard Test Access port (IEEE Std. 1149.1), for the full state diagram.

The main state diagram consists of six steady states: Test-Logic-Reset, Run-Test/Idle, Shift-DR, Pause-DR, Shift-IR, and Pause-IR. A unique feature of this protocol is that only one steady state exists for the condition when jtag_tms is set high: the Test-Logic-Reset state. This means that a reset of the test logic can be achieved within five jtag_tck(s) or less by setting the jtag_tms input high.

At power up or during normal operation of the processor, the TAP is forced into the Test-Logic-Reset state by driving jtag_tms high and applying five or more jtag_tck(s). In this state, the TAP issues a reset signal that places all test logic in a condition that does not impede normal operation of the processor. When test access is required, a protocol is applied via the jtag_tms and jtag_tck inputs, causing the TAP to exit the Test-Logic-Reset state and move through the appropriate states. From the Run-Test/Idle state, an Instruction Register scan or a Data Register scan can be issued to transition the TAP through the appropriate states shown in Figure 21.5.

The states of the Data and Instruction Register scan blocks are mirror images of each other adding symmetry to the protocol sequences. The first action that occurs when either block is entered is a capture operation. For the Data Registers, the Capture-DR state is used to capture (or parallel load) the data into the selected serial data path. In the Instruction Register, the Capture-IR state is used to capture status information into the Instruction Register.

From the Capture state, the TAP transitions to either the Shift or Exit1 state. Normally the Shift state follows the Capture state so that test data or status information can be shifted out for inspection and new data shifted in. Following the Shift state, the TAP either returns to the Run-Test/Idle state via the Exit1 and Update states or enters the Pause state via Exit1. The reason for entering the Pause state is to temporarily suspend the shifting of data through either the Data or Instruction Register while a required operation, such as refilling a host memory buffer, is performed. From the Pause state shifting can resume by re-entering the Shift state via the Exit2 state or terminated by entering the Run-Test/Idle state via the Exit2 and Update states.

Notes

Upon entering the Data or Instruction Register scan blocks, shadow latches in the selected scan path are forced to hold their present state during the Capture and Shift operations. The data being shifted into the selected scan path is not output through the shadow latch until the TAP enters the Update-DR or Update-IR state. The Update state causes the shadow latches to update (or parallel load) with the new data that has been shifted into the selected scan path. Limitations of TAP controller are RC32300 CPU core as part of the RC32334.

TAP Controller State Assignments

All state transitions within the TAP controller occur at the rising edge of the TCLK pulse and—depending on the jtag_tms signal level (0 or 1)—it proceeds to the next state.

- *Test-Logic-Reset*
  - *The test logic is disabled so that normal operation of the on-chip system logic can continue unhindered.*
- *Run-Test/Idle*
  - *A controller state between scan operations.*
- *Select-DR-Scan*
  - *This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state.*
- *Capture-DR*
  - *In this controller state, data may be parallel-loaded into test data registers selected by the current instruction on the riding edge of TCLK.*
- *Shift-DR*
  - *In this controller state, the test data register connected between jtag_tdi and jtag_tdo, as a result of the current instruction, shifts data one stage towards its serial output on each rising edge of TCLK. The test data register content is being shifted out serially, LSB first, at the falling edge of TCLK towards the jtag_tdo output.*
- *Exit1-DR*
  - *This is a temporary controller state. If jtag_tms is held high, a rising edge applied to TCLK while in this state causes the controller to enter the Update-DR state, which terminates the scanning process. If jtag_tms is held low and a rising edge is applied to TCLK, the controller enters the Pause-DR state.*
- *Pause-DR*
  - *This controller state allows shifting of the test data register in the serial path between jtag_tdi and jtag_tdo to be temporarily halted.*
- *Exit2-DR*
  - *This is a temporary controller state. If jtag_tms is held high and a rising edge is applied to TCLK while in this state, the scanning process terminates and the TAP controller enters the Update-DR state.*
- *Update-DR*
  - *Data is latched onto the parallel output of these test data registers from the shift-register path on the falling edge of TCLK.*
- *Select-IR-Scan*
  - *This is a temporary controller state in which all test data registers selected by the current instruction retain their previous state.*
- *Capture-IR*
  - *In this controller state, the shift-register contained in the instruction register loads a pattern of fixed logic values on the rising edge to TCLK.*
- *Shift-IR*
  - *In this controller state, the shift-register contained in the instruction register is connected between jtag_tdi and jtag_tdo and shifts data one stage towards its serial output on each rising edge to TCLK. The instruction shift register content is being shifted out serially, LSB first, at the falling edge of TCLK towards the jtag_tdo output.*

Notes

- *Exit1-IR*
  - *This is a temporary controller state. While in this state, if jtag_tms is held high, a rising edge applied to TCLK causes the controller to enter the Update-DR state, which terminates the scanning process. If jtag_tms is held low and a rising edge is applied to TCLK, the controller enters the Pause-DR state.*
- *Pause-IR*
  - *This controller state allows shifting of the instruction register to be halted temporarily.*
- *Exit2-IR*
  - *This is a temporary controller state. While in this state, if jtag_tms is held high and rising edge is applied to TCLK termination of the scanning process occurs. The TAP controller then enters the Update-IR controller state. If jtag_tms is held low and a rising edge is applied to TCLK, the controller enters the Shift-IR state.*
- *Update-IR*
  - *The instruction shifted into the instruction register is latched to the parallel output from the shift-register path on the falling edge of TCLK, in this controller state. Once the new instruction has been latched, it becomes the current instruction.*

Instruction Register (IR)

The Instruction Register is responsible for providing the address and control signals required to access a particular Data Register in the scan path. The Instruction Register is accessed when the TAP receives an Instruction Register scan protocol. During an Instruction Register scan operation, the TAP controller selects the output of the Instruction Register to drive the TDO pin. The Instruction Register consists of an instruction shift register and an instruction shadow latch. The instruction shift register consists of a series of shift register bits arranged to form a single scan path between TDI and TDO. During Instruction Register scan operations, the TAP controls the instruction shift register to capture status information and shift data from TDI to TDO. Both the capture and shift operations occur on the rising edge of TCK; however, the data shifted out from the TDO occurs on the falling edge of TCK. The status inputs are user-defined observability inputs, except for the two least significant bits, which are always 01 for scan-path testing purposes. (The Instruction Register has a minimum length of two bits.) In the Test-Logic-Reset state, the instruction shift register is set to all ones. This forces the device into the functional mode and selects the Bypass Register (or the Device Identification Register if one is present).

The instruction shadow register consists of a series of latches, one latch for each instruction shift register bit. During an Instruction Register scan operation, the latches remain in their present state. At the end of the Instruction Register scan operation, the Instruction Register update input updates the latches with the new instruction installed in the instruction shift register. In the Test-Logic-Reset state, the latches are set to all ones.

Test Data Register (DR)

The IEEE 1149.1 standard requires two Data Registers: Boundary-Scan Register and Bypass Register, with a third, optional, Device Identification Register. Additional user-defined Data Registers may be included. The Data Registers are arranged in parallel from the primary TDI input to the primary TDO output. The Instruction Register supplies the address that allows one of the Data Registers to be accessed during a Data Register scan operation. During a Data Register scan operation, the addressed scan register receives TAP control signals to pre-load test response and shift data from TDI to TDO. During a Data Register scan operation, the TAP selects the output of the Data Register to drive the TDO pin. When one scan path in the Data Register is being accessed, all other scan paths remain in their present state.

However, additional specific test data registers are available for various operations during Run-Time and Real-Time debugging. These registers are connected in parallel between a common serial input and a common serial data output.

The following sections provide a brief description of these elements. For a complete description, refer to IEEE Standard Test Access port (IEEE Std. 1149.1 - 1990).

Notes

Bypass Register

The Bypass Register is used to allow test data to flow through the device from TDI to TDO. It contains a single-stage shift register for a minimum length in serial path. When an instruction selects the bypass register and the TAP controller is in the Capture-DR state, the shift register stage is set to a logic zero on the rising edge of TCLK. Bypass register operations should not have any effect on the device's operation in response to the BYPASS instruction.

Boundary Scan Register

The Boundary Scan Register allows serial data to be loaded into or read out of the processor input/output ports. The Boundary Scan Register is a part of the IEEE 1149.1 - 1990 Standard JTAG Implementation. The boundary scan register for the internal CPU core must never be used.

Device Identification Register

The Device Identification Register is an optional register defined by IEEE 1149.1, to identify the device's manufacturer, part number, revision, and other device-specific information. Table 21.2 shows the bit assignments defined for the (read only) Device Identification Register. These bits can be scanned out of the Identification Register after being selected. Although the Device Identification Register is optional, IEEE 1149.1 specification has dedicated an instruction to select this register. The Device Identification Register is selected when the Instruction Register is loaded with the IDCODE instruction.

Bit(s)	Mnemonic	Description	R/W	Reset
0	reserved	reserved 0x1	R	1
11:1	Manuf_ID	**Manufacturer Identity** (11 bits) IDT 0x33	R	0x33
27:12	Part_number	**Part Number** (16 bits) This field identifies the part number of the processor derivative. For the RC32300 CPU core, this value is: 0x0026	R	impl. dep.
31:28	Version	**Version** (4 bits) This field identifies the version number of the processor derivative. For the RC32300 CPU core, this value is 0x0	R	impl. dep.

Table 21.2 CPU Core Device Identification Register

.

Version	Part Number	Vendor ID	LSB
0000	0000|0000|0010|0110	0000|0110|011	1

Figure 21.6 CPU Core Device ID Instruction Format

Implementation Register

This is a 32-bit read only register to identify the features of the Debug Support Unit which are implemented by the RC32334.

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
0	MIPS32/64	**MIPS 32-bit or 64** indicates the type of MIPS CPU, informing the width of the MIPS CPU datapath, the debug registers implemented in the DSU, and the EJTAG_Data_register. 0: 32-bit wide data registers 1: 64-bit wide data registers **Set to 0 for RC32334**	R	0
4..1	Ch[3:0]	**Number of Break Channels**: these 4 bits used to indicate the number of break channels implemented in the DSU. Debug SW should check InstrBrk, DataBrk and ProcBrk to know what break types are implemented. 0000 = no break channels 0001 = 1 break channel (Default) ... 1111 = 15 break channels	R	0001
5	NoInstBrk	**Instruction Address Break**: this bit indicates if the Instruction Address Break function is implemented in the DSU. 0: Instruction Address Break is implemented 1: Instruction Address Break is not implemented	R	0
6	NoDataBrk	**Data Address Break**: this bit indicates if the Data Address Break function is implemented in the DSU. 0: Data Address Break is implemented 1: Data Address Break is not implemented	R	0
7	NoProcBrk	**Processor Bus Break**: this bit indicates if the Processor Bus Break function is implemented in the DSU. 0: Processor Bus Break is implemented 1: Processor Bus Break is not implemented	R	0
10..8	PCSTW	**PCST Width and DCLK Division Factor** 000,111 3 bits (DCLK is 1/1 of CPU pipeline CLK) 001 6 bits (DCLK is 1/2 of CPU pipeline CLK) 010 9 bits (DCLK is 1/3 of CPU pipeline CLK) 011 12 bits (DCLK is 1/4 of CPU pipeline CLK) others reserved	R	000
13..11	TPCW	**TPC Width** 000,111 1 bit 000 is Standard EJTAG 001 2 bits Reserved 010 4 bits Reserved 011 8 bits Reserved others reserved	R	000
14	NoDMA	No EJTAG DMA Support 0: EJTAG DMA is supported by implementation 1: EJTAG DMA is not supported by implementation	R	1
15	NoPCTrace	No PC Trace Support 0: PC Trace is supported by implementation 1: PC Trace is not supported by implementation	R	0
16	MIPS16	MIPS16 Support 0: MIPS CPU does not support MIPS16 1: MIPS CPU supports MIPS16	R	0
17	ICacheC	Instruction Cache Coherency 0: Instruction Cache does not keep DMA coherency 1: Instruction Cache keeps coherency with DMA	R	0
18	DCacheC	Data Cache Coherency 0: Data Cache does not keep coherency with DMA 1: Data Cache keeps coherency with DMA	R	0

Table 21.3 Implementation Register (Part 1 of 2)

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
19	PhysAW	Physical Address Width Informs the size of EJTAG_Address_register 0: Physical addresses are 32-bit in length 1: Physical addresses is from 33 to 64-bits in length The exact length of can be determined by shifting a pattern through the EJTAG Address Register.	R	0
22..20	reserved	**reserved,** EJTAG Probe must shift 0s in	R	0
23	SDBBPCode	SDBBP uses Special2 Opcode (for MIPS-I/II/II/IV) 0: SDBBP is encoded according to EJTAG rw 1.3 specification 1: SDBBP is encoded using a Special2 Opcode	R	0
31..24	reserved	**reserved,** EJTAG Probe must shift 0s in	R	0

Table 21.3 Implementation Register (Part 2 of 2)

EJTAG Address Register

The length of the EJTAG Address Register is 32 bits. The length is identical to the length of the physical processor address bus, and is determined by shifting a pattern through the register. This register can be used as follows:

- *Processor Access: In this mode the RC32334 can access memory on the EJTAG Probe in a serial way through the JTAG interface. A 32 bit address is captured and is shifted out via the jtag_tdo/ ejtag_tpc pin to the EJTAG Probe. Depending on the direction of the access, data is shifted into the jtag_tdi pin (processor read) or shifted out of the jtag_tdo/ejtag_tpc pin (processor write).*

EJTAG Data Register

This register is used with the EJTAG_Address_register in the following mode:

- *Processor Access: In this mode the RC32334 can access memory located on the EJTAG Probe in a serial way through the JTAG interface. A 32 bit data word is captured and is shifted out via the jtag_tdo/ejtag_tpc pin to the EJTAG Probe for a Processor Write action; for a Processor Read action 32 bits of data is shifted into the jtag_tdi /ejtag_dint_n* pin and is made available to the processor.*

The organization of the bytes in the 32 bit EJTAG Data Register depends on the endianess of the CPU, as shown in Figure 21.7 and Figure 21.8.

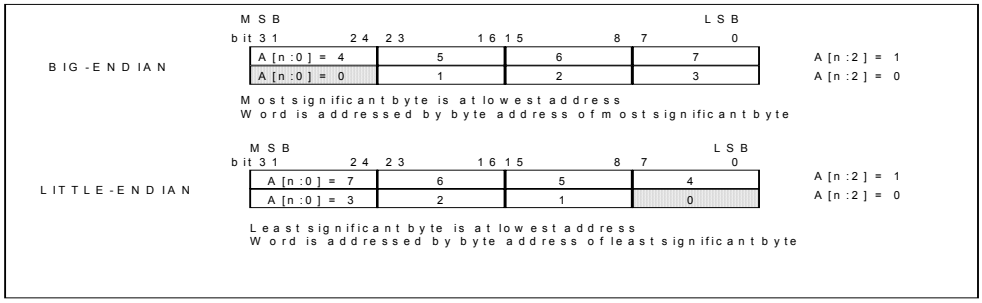


Figure 21.7 Byte Organization in a 32-bit EJTAG Data Register

Notes

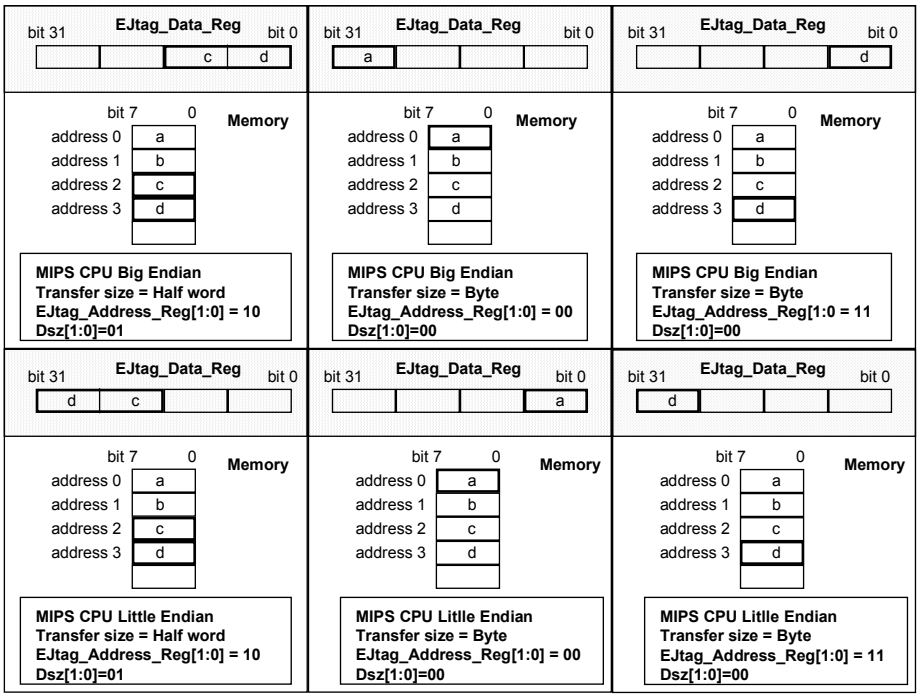


Figure 21.8 Examples of Byte Organization in a 32-bit EJTAG Data Register

EJTAG Control Register

This is a 32 bit register to control the various operations of the debug support and the JTAG unit. This register is selected by shifting in the JTAG_CONTROL_IR instruction. Bits in the EJTAG_Control_register can be set/cleared by shifting in data; status is read by shifting out this register.

This EJTAG_Control_register, shown in Table 21.4, can only be accessed by the JTAG interface.

Bit(s)	Mnemonic	Description	R/W	Reset
0	ClkEn	**DCLK output Enable bit** When this bit is set to 0 it disables the DCLK output (making it high impedance or 0). When it is set to 1 it will enable the DCLK output during Real Time Tracing mode.	R/W	0
1		**Unused** This bit is always 0.	W0/R	0
2		**Unused** This bit is always 0.	W1/R	0
3	BrkSt	**Break Status** This bit is set to 1 when the processor takes a debug exception and is cleared when the processor executes the DERET instruction. This bit is the same as the DM (Debug Mode) bit in Debug Register [30].	R	0
4		**Unused** This bit is always 0.	R/W	0
5		**Unused** This bit is always 0.	R/W	0
6		reserved	R	0
8,7	Dsz[1:0]	**Unused** These bits are always 00.	R/W	00

Table 21.4 EJTAG_Control_Register (Part 1 of 3)

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
9		**Unused** This bit is always 0.	R/W	0
10		**Unused** This bit is always 0.	R	0
11		**Unused** This bit is always 0.	W1/R	0
12	JtagBrk	**JTAG Break** Setting this bit to 1 causes a debug exception to the processor. This bit is also set by activating the jtag_tdi/ejtag_dint_n* pin (stopping the PC Trace mode). When the debug exception occurs, the processor core will be waken up if it was in sleep mode. This bit is cleared by hardware when the debug exception is taken. JTAG Break is ignored if the CPU is in debug mode.	W1/R	0
14..13	reserved	**reserved**	R	00
15	ProbEn	**EJTAG Probe Enable** This bit must be set to 1 by the probe's software to indicate that a probe is present and active. If it is set to 0, it indicates that the probe is not present or inactive and the EJTAG module will not allow the processor to access the probe, and the result is undefined (may result in bus error). The clock at the DCLK pin is disabled in this case. The debug exception is set at 0xBFC0-0480.	R/W	0
16	PrRst	**Processor Reset** When this bit is set to 1, a soft reset exception is forced to the processor. The reset is sustained as long as the PrRst bit is 1. The processor will set the SR bit in the processor's status register. The Processor Reset bit is not masked by MRst* in Debug Control Register[1].	R/W	0
17		**Unused** This bit is always 0.	R/W	0
18	PrAcc	**Processor Access** This bit is set to 1 by hardware when the processor accesses the probe's reserved addresses (0xFF20-0000 through 0xFF2F-FFFF). The probe's software must set this bit to 0 to indicate the end of the access action.	W0/R	0
19	PRnW	**Processor Access Read not Write** Internal hardware sets this bit to 1 when the Processor Access action is a write action, it is set to 0 for a read action.	R	0
20	PerRst	**Peripheral Reset** When this bit is set to 1 it will force a reset to all the peripherals of the processor (except for the EJTAG interface and the Debug Support Unit).	R/W	0
21	Run	**Run** When this bit is read as 1, the processor was in the run state (the processor clock was running) at the moment that the EJTAG_Control_register was captured.	R	1
22		**Unused** This bit is always 0.	R	0

Table 21.4 EJTAG_Control_Register (Part 2 of 3)

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
23	Sync	**Sync** *(only used when PCTRACE is supported)* This bit will synchronize the end of the Processor read access (instruction fetch) with the setting of the PC Trace mode. When this bit is set, the processor will be stalled for the last processor read access until the EJTAG module has been placed into the PC Trace Mode (i.e. the PC Trace instruction is in the instruction register and the TAP controller is in the Run-Test/idle state). This bit can only be set at the end of a processor read access, i.e. the PrAcc bit was 1 and is written with a 0 and PRnW is 0. In all other cases, writing a 1 will be ignored. The bit is cleared by hardware when the PC Trace mode is entered. The bit can also be cleared by writing a 0 to it: this will then also generate the acknowledge for the processor read. This bit is read-only 0 when PC Trace is not supported (NoPCTrace = 1).	W/R	0
25..24	PCLen	**Target PC Output Length** Set to 00 for RC32334.	R	0
31..26	reserved	**reserved**	R	0

Table 21.4 EJTAG_Control_Register (Part 3 of 3)

Figure 21.9 shows examples of the Sync Operation.

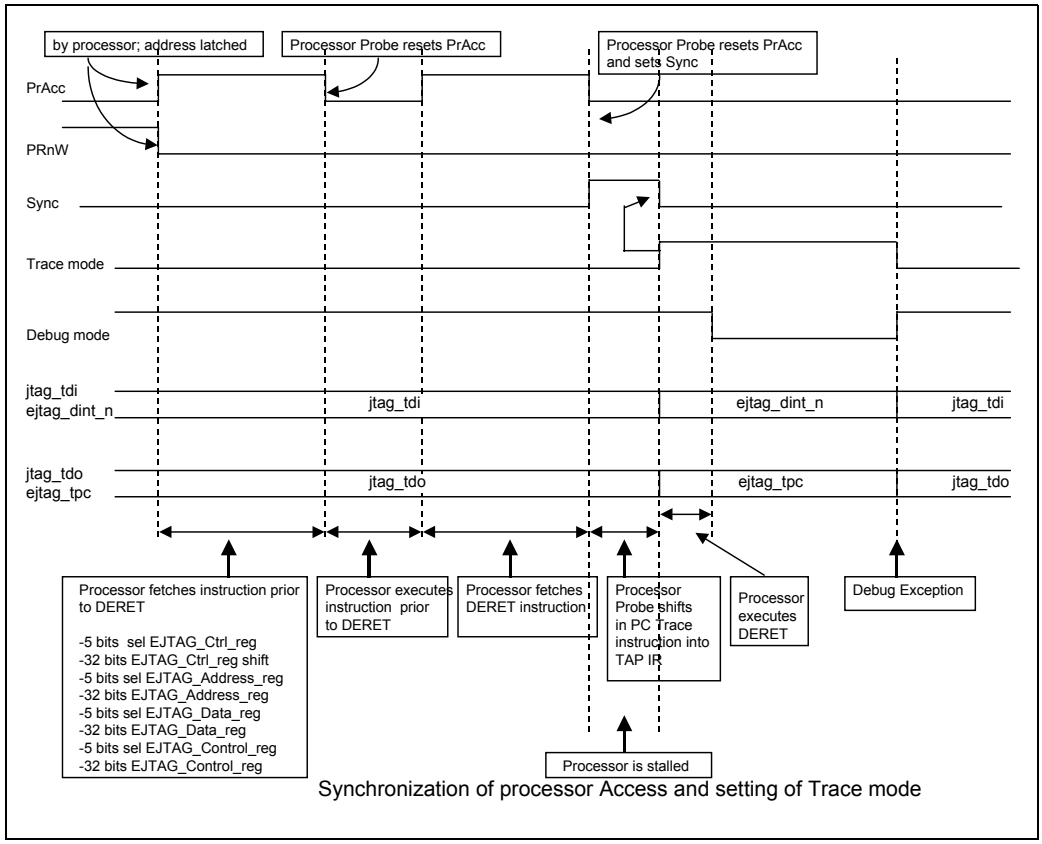


Figure 21.9 Examples of the Sync Operation

PC Trace Instruction (only if PC Trace is supported)

This JTAG instruction is used to enable PC Trace mode. The Real-Time Trace mode is set when the TAP controller has reached the Run-Test/Idle state. In this mode, the jtag_tdo/ejtag_tpc pin provides non-sequential program counter output at the ejtag_dclk speed. The ejtag_pcst[2:0] pins are used to show the type of instruction execution. A debug exception disables the PC Trace mode. The instruction register will be set to BYPASS code (0x1F).

Notes

Processor Access

The CPU can then execute code taken from the EJTAG Probe and it can access data (via load or store) which is located on the EJTAG Probe. This occurs in a serial way through the EJTAG interface: the core can thus execute instructions e.g. debug monitor code, without occupying the user's memory.

Accessing the EJTAG Probe's memory can only be done when the processor accesses an EJTAG address (which is in the range from 0xFF20-0000 to 0xFF2F-FFFF), when the ProbEn bit is set and when the processor is in debug mode (DM=1).

When a debug exception is taken, while the ProbEn bit is set, the processor will start fetching instructions from address 0xFF20-0200.

Instruction Fetch/Read from the EJTAG Probe

1. The internal hardware latches the requested address into the JTAG_Address_Capture Register (in case of the Debug exception: 0xFF20-0200).
2. The internal hardware sets the following bits in the EJTAG_Control_register:
   PrAcc = 1 (selects Processor Access operation)
   PRnW = 0 (selects processor read operation)
   Dsz[1:0] = value depending on the transfer size
3. The EJTAG Probe selects the EJTAG_Control_register, shifts out this control register's data and tests the PrAcc status bit (Processor Access): when the PrAcc bit is found 1, it means that the requested address is available and can be shifted out.
4. The EJTAG Probe checks the PRnW bit to determine the required access and shifts in a DmaAcc = 0 bit into the EJTAG_Control_register.
5. The EJTAG Probe selects the EJTAG_Address_register and shifts out the requested address.
6. The EJTAG Probe selects the EJTAG_Data_register and shifts in the instruction corresponding to this address.
7. The EJTAG Probe selects the EJTAG_Control_register and shifts a PrAcc = 0 bit into this register to indicate to the processor that the instruction is available.
8. The instruction becomes available in the instruction register and the processor starts executing.
9. The processor increments the program counter and outputs an instruction read request for the next instruction. This will start the whole sequence again.

Using the same protocol, the processor can also execute a load instruction to access the EJTAG Probe's memory. For this to happen, the processor must execute e.g. a lw, lb,... instruction with the target address in the appropriate range.

Almost the same protocol is used to execute a store instruction to the EJTAG Probe's memory. The store address must be in the range: 0xFF20-0000 to 0xFF2F-FFFF, the ProbEn bit must be set, and the processor has to be in debug mode (DM=1). The sequence of actions is found below.

Processor Write Access

1. The internal hardware latches the requested address into the JTAG_Address_Capture Register
2. The internal hardware latches the data to be written into the JTAG_Data_Capture Register.
3. The internal hardware sets the following bits in the EJTAG_Control_register:
   PrAcc = 1 (selects Processor Access operation)
   PRnW = 1 (selects processor write operation)
   Dsz[1:0] = value depending on the transfer size
4. The EJTAG Probe selects the EJTAG_Control_register, shifts out this control register's data and tests the PrAcc status bit (Processor Access): when the PrAcc bit is found 1, it means that the requested address is available and can be shifted out.
5. The EJTAG Probe checks the PRnW bit to determine the required access and shifts in a DmaAcc=0 bit into the EJTAG_Control_register.
6. The EJTAG Probe selects the EJTAG_Address_register and shifts out the requested address.
7. The EJTAG Probe selects the EJTAG_Data_register and shifts out the data to be written.
8. The EJTAG Probe selects the EJTAG_Control_register and shifts a PrAcc = 0 bit into this register to indicate to the processor that the write access is finished.
9. The EJTAG Probe writes the data to the requested address in its memory.
10. The processor detects that PrAcc bit = 0, which means that it is ready to handle a new access.

Notes

Figure 21.10 depicts the processor and probe actions for the Processor Read and Processor Write Access.

.

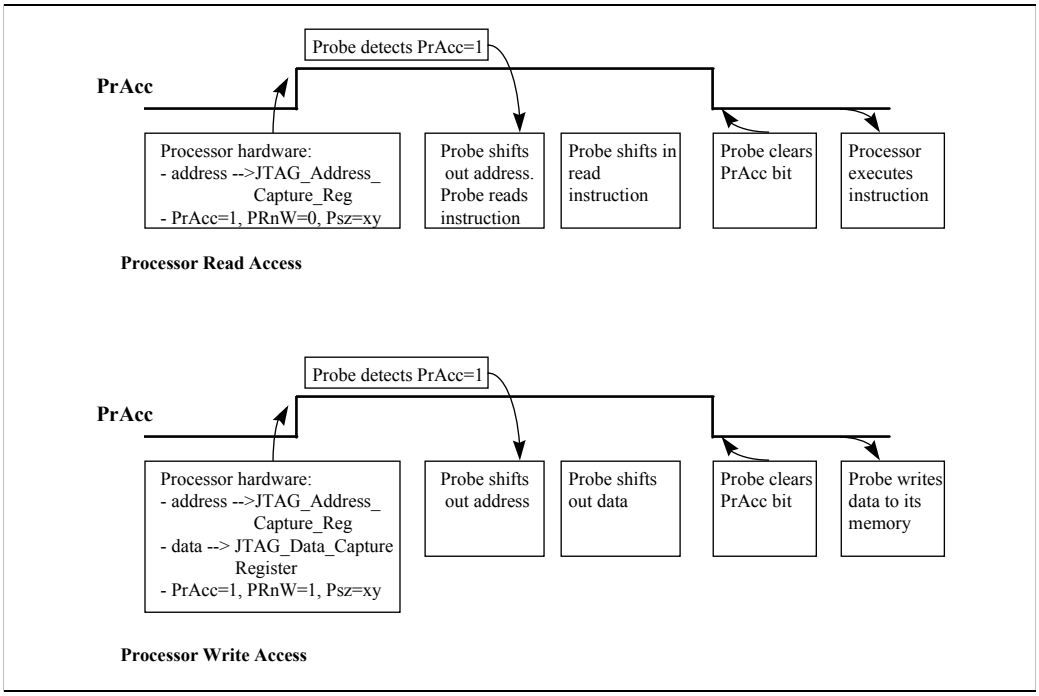


Figure 21.10 EJTAG Processor Access

Reset Overview

The processor core, processor peripherals, EJTAG module and the DSU can be reset as follows (see also Figure 21.11):

- *The hard reset (general reset) signal resets the processor, the EJTAG, the DSU and the peripherals.*
- *The EJTAG Probe can Soft Reset the processor core by setting the PrRst bit in the EJTAG_Control_register.*
- *The EJTAG Probe can reset the peripherals on the processor by setting the PerRst bit in the EJTAG_Control_register.*
- *The processor can reset both the EJTAG Module and the DSU by setting the JtagRst bit in the Debug Register.*

A System reset can be provided by the EJTAG Probe by activating the combination. of reset control bits: PrRst and PerRst.

A full system reset through the EJTAG is by the JTAG reset pin to the master reset on the board.

Notes

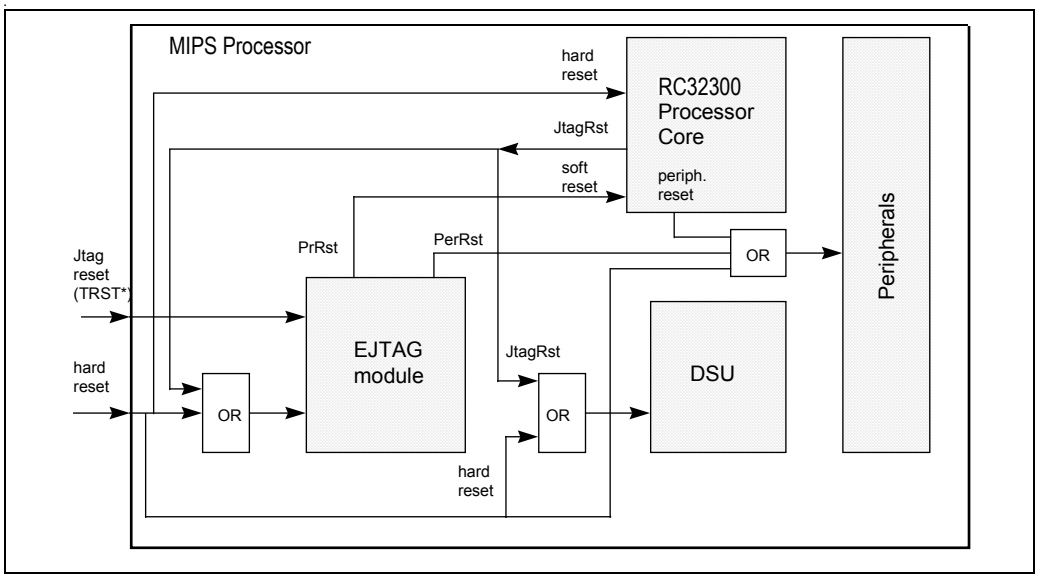


Figure 21.11 Reset Overview

EJTAG Module Clocking

The bus clock may be used to clock all registers within the EJTAG Module which are not part of the TAP-controller or the JTAG registers (e.g. used for Processor Access or DMA access). These latter registers are clocked by jtag_tck.

Instruction Register

The instruction Register is a 5-bit field (such as IR4, IR3, IR2, IR1, IR0) that is used to decode 32 different possible instructions and allows instructions to be serially input to the device, when the TAP controller is in the Shift-IR state.

Instructions are decoded to perform the following tasks:

- *To select test data registers that may operate while the instruction is current. The other test data registers should not interfere with chip operation and selected data registers.*
- *To define the serial test data register path that is used to shift data between TDI and TDO during data register scanning.*

Instructions are decoded as shown in Table 21.5. Brief descriptions of each instruction are included in the table, but for a more complete description, refer to IEEE Standard Test Access port (IEEE Std. 1149.1-1990).

Hex Value	Instruction Name/Description	Function
0x00	**EXTEST** Extest is a mandatory instruction provided for external circuitry and board level interconnection check.	Select Boundary Scan Register.
0x01	**IDCODE** Selects the Device Identification Register to read out manufacture's identity, part number, and version number.	Select Chip Identification Data Register.
0x02	**SAMPLE/PRELOAD** SAMPLE instruction allows a snapshot of data flowing from the system pins to the on-chip logic, or vice versa. Preload allows data values to be loaded onto the latched parallel outputs of the boundary-scan shift register, prior to selection of the other boundary-scan test instruction.	Select Boundary Scan Register.

Table 21.5 Instruction Decoding (Part 1 of 2)

Notes

Hex Value	Instruction Name/Description	Function
0x03	**ImpCode**	Select Implementation Register.
0x04	**INTEST** Tests the processor's internal logic. Test simulations are shifted in one at a time and applied to the on-chip system logic. The test results are captured into the Boundary-scan register and examined by subsequent shifting.	JTAG
0x05	**HI-Z** Places all of the device's output pins into a high impedance state. An external ICE can then drive all of the pins, and would not damage on-chip logic as well as the input pins.	JTAG
0x06	**CLAMP** Allows the state of the signals driven from the IC pins to be determined from the boundary-scan register, while the bypass register is selected as the serial path between TDI and TDO.	JTAG
0x07	**BYPASS**	Bypass mode
0x08	**JTAG_ADDRESS_IR** Selects the JTAG_Address_Register from external ICE probe to load 32-bits of TDI data into the JTAG_Address_Register. At the DR-Update moment, the shifting stops and the 320bits of data are then loaded into the update register for the internal bus.	Select JTAG_Address Register.
0x09	**JTAG_DATA_IR** Selects the JTAG_Data_Register from external ICE probe to load 32-bits of TDI data into the JTAG_Data_Register. In addition, data written to external ICE probe are captured from the processor or any slave at Data_Capture register. Data latched are at Capture_DR stage are shifted out via TDO.	Select JTAG_Data Register.
0x0A	**JTAG_CONTROL_IR** Select the JTAG_Control_Register from the external ICE probe, to load 32-bits of TDI data into JTAG_Control_Register bits or read the JTAG_Control_Register bits.	Select JTAG_Control register
0x0B	**JTAG_ALL_IR** This register is the concatenation of the Address_Shift, Data_Shift and JTAG_Contrl_Register. It can be used if switching instructions in the instruction register cost too many TCLK cycles. The first bit shifted out is bit 0 as shown in Figure 21.12	Select JTAG_All register
0x10	**PCTRACE** Decoded to switch from Run-Time mode to Real-Time mode. After executing this instruction, the PCST[2:0] pins, in conjunction with TDO, provides a non-sequential program counter at the DCLK speed. TDI/DINT* is used in Real-time mode to switch back to Run-Time Mode by setting the JtagBrk bit. The instruction register will be set to BYPASS code (0x1F). Prior to executing the PCTRACE instruction, the TAP controller is placed into the Run-Test/Idle state.	PCTRACE Instruction
0x1F	**BYPASS** Contains a single shift-register stage and is set to provide a minimum-length serial path between TDI and the TDO pins of the device, when no device test operations are required. Any unused instruction is defaulted to the BYPASS instruction.	Bypass mode

Table 21.5 Instruction Decoding (Part 2 of 2)

Notes

Note: As mentioned in the definition of the BYPASS instruction, any unused instruction will default to the BYPASS instruction.

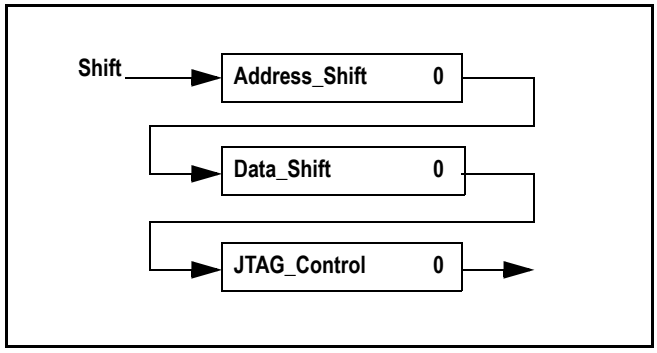


Figure 21.12 Shift Order Sequence of the JTAG_All_IR Register

The Debug Unit

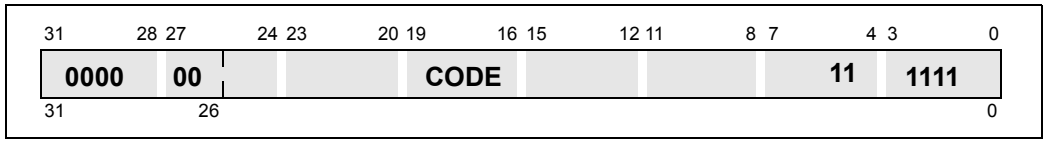
The Debug Unit section describes the debug unit implemented in the processor, and covers the extended instruction to MIPS ISA instruction set as well as support functions and registers for Real Time Debugging.

Note: The EXTERNAL INSTRUCTIONS are slightly different from the original definition. Similarly, the DEBUG REGISTER is also different.

Extended Instructions

The following instructions are added to the standard MIPS ISA instruction set to provide a software debug breakpoint exception and debug exception return.

SDBBP (Software Debug Breakpoint)



Format **SDBBP Code**

Description This instruction raises a Debug Breakpoint exception, passing control to an exception handler. The code field can be used for passing information to the exception handler, but the only way to have the code field retrieved by the exception handler is to load the contents of the memory word containing this instruction, using the DEPC register. The SDBBP instruction is NOP when it is used in debug mode (DM='1'). The CODE field of the SDBBP is available for use as a software parameter only, and is retrieved by the debug exception handler only by loading the contents of the memory work containing the instruction. The CODE field is not used in any way by the hardware.

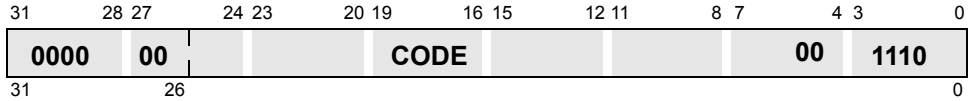
Operation

```
T: IF not in Debug Mode
   PC <– ExceptionHandlerVector
   if DBD = '0', DEPC <- Address of SDBBP instruction
   else DEPC <- Address of branch (taken) instruction
   DM <- '1'
   BrkSt, DBp <- '1'
ELSE NOP
```

Exceptions Debug Breakpoint Exception

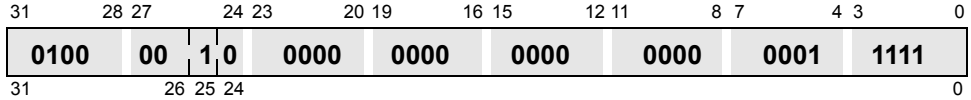
Note: The RC32334 implements the following opcode:

Notes



Description The opcode above was used by MIPS CPUs following EJTAG specification 1.3.1, and its use is discouraged because it may conflict with a future MIPS ISA.

DERET (Debug Exception Return)



Format **DERET**

Description This instruction executes a return from a debug exception. It has a branch delay slot, the same as the branch or jump instruction cycle, executing with a delay of one instruction cycle. The DERET instruction can not be used in the delay slot itself. The return address stored in the DEPC register is copied to the PC and processing returns to the original program. The Debug Mode bit (DM in Debug [30]) and the BrkSt bit (EJTAG_Control_Register[3]) are reset.
Note: If a MTC0 instruction was used to set the return address in the DEPC register, a minimum of two instructions must be executed before executing the DERET.

Operation
```
T: temp <- DEPC
T+1: PC <- temp
   DM <- '0'
   BrkSt <- '0'
```

Exceptions Coprocessor Unusable Exception

Extended CP0 Registers (Debug Registers)

The Standard EJTAG Specification (Version 1.5.3) defines three registers to be added to the CP0 registers to support debug exceptions:

- *Debug Register*
- *Debug Exception PC*
- *Debug Exception Save Register*

The RC32334 only implements the Debug register and the Debug Exception PC register in the CP0. The Debug Exception Save Register is implemented as an on-chip register located at physical address 0xFFFF-E210.

Debug Register

Debug Register, CPO register 24

The Debug Register is used to control the debug exception and provide status information about the cause of the debug exception. The read only status bits are automatically updated every time the debug exception is taken.

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
0		**Unused** This bit is always 0.	R	0
1	DBp	**Debug Breakpoint exception status** This bit is set to '1' when a debug exception occurred due to execution of the SDBBP instruction.	R	0
2	DDBL	**Debug Data Address Break Load Exception Status** This bit is set to '1' when a Data Address Break caused the debug exception during execution of a Load Memory instruction.	R	0
3	DDBS	**Debug Data Address Break Store Exception Status** This bit is set to '1' when a Data Address Break caused the debug exception during execution of a Store Memory instruction.	R	0
4	DIB	**Debug Instruction Address Break Exception Status** This bit is set to '1' when an Instruction Address Break caused the debug exception.	R	0
5	DINT	**Debug Processor Bus Break Exception Status** This bit is set to '1' when a Processor Bus Break or a JTAG Break (from the EJTAG Probe) caused the debug exception.	R	0
6	DBES	Debug Boot Bit This bit is set to '1' when Debug Boot is active during reset and forces the CPU to take the Debug Exception at the end of the reset sequence. It is cleared by software.	W0/ R	0
7	JtagRst	**JTAG Reset** Setting this bit to '1' will reset both the EJTAG module and the DSU.	R/W	0
8		**Unused** This bit is always 0.	R	0
9	reserved	reserved	R	0
10	BsF	**Bus Error Exception Flag** This bit is set to '1' when a bus error exception occurred during a Load or Store instruction while the debug exception handler was running (DM='1'). The Bus error Exception will set this bit to '1' regardless of writing a '0'. It is cleared by writing a '0' and writing '1' is ignored.	W0/R	0
11	TLF	**TLB Exception Flag** This bit is set to '1' when a TLB related exception occurs during the Load or Store instruction while the debug exception handler is running (DM='1'). The TLB exception will set this bit to '1', and it is cleared by writing '0'. Writing '1' is ignored.	W0/R	0
12	OES	**Other Exception Status** When this bit is set it indicates an exception other than Reset, cache error, NMI or UTLB Miss/TLB Refill was raised at the same time as a debug exception. In this case the Status, Cause, EPC and BadVaddr registers assume the usual status after occurrence of such an exception, but the address in the DEPC is not the 'other exception' vector address. In this case the proper exception handler address has to be placed in DEPC by the debug exception handler software, after which processing returns directly from the debug exception to the other exception handler.	R	0
13	TRS	**TLB Refill Miss Status** This bit is the same as OES, but it is set when TLB refill occurs at the same time as a debug exception. DEPC must be set to TLB Refill exception vector, that is, 0xBFC0_0200 (BEV=1) or 0x8000_0000 (BEV=0), by debug exception handler software, after which processing returns directly from the debug exception to the TLB Refill exception handler. **For a description of the exception vector locations for the RC32334, refer to the Exception Vector Locations section in Chapter 6 (Table 6.16) of this manual.**	R	0

Table 21.6 Debug Register (Part 1 of 2)

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
14	NIS	**Non Maskable Interrupt Status** When this bit is set it indicates that a non-maskable interrupt has occurred at the same time as a debug exception. In this case the Status, Cause, EPC and BadVAddr registers assume the usual status after occurrence of a non-maskable interrupt, but the address in DEPC is not the non-maskable exception vector address (0xBFC0-0000). Instead, 0xBFC0-0000 has to be placed in DEPC by the debug exception handler software, after which processing returns directly from the debug exception to the non-maskable interrupt handler.	R	0
15	CES	Cache Error Status: This bit indicates that a Debug exception and a Cache Error occurred at the same time. 0: No Cache Error. 1; Cache Error occurred at the same time with Debug exception.	R	0
16	Itrpt	Interrupt when Cause.IV is set: This bit indicates that a Debug exception and an interrupt with the Cause.IVbit set occurred at the same time. 0: No interrupt with Cause.IV bit set. 1; Interrupt with Cause.IV bit set.	R	0
17-29	reserved	reserved	R	0
30	DM	**Debug Mode Status** When this bit is set it indicates that a debug exception has been taken. It is cleared upon return from the debug exception (execution of DERET). While this bit is set all interrupts (including NMI), TLB exception, Bus error exception and debug exception are masked and the cache line locking function is disabled. A copy of the DM status is available in the BrkSt bit (EJTAG_Control_Register[3]) and the ejtag_pcst[2:0] status lines (DBM code).	R	0
31	DBD	**Debug Branch Delay** This bit is set to '1' when a debug exception occurs while an instruction in the branch delay slot is executing. The DEPC points to the branch or jump instruction preceding the instruction causing the debug exception.	R	0

Table 21.6 Debug Register (Part 2 of 2)

Debug Exception Program Counter Register (DEPC)

For the RC32334, DEPC is CP0 Register 23.

.

Bit(s)	Mnemonic	Description	R/W	Reset
31:0	DEPC	The DEPC register holds the address where processing resumes after the debug exception routine has finished. The address that has been loaded in the DEPC register is the virtual address of the instruction that caused the debug exception. If the instruction is in the branch delay slot, the virtual address of the immediately preceding branch or jump instruction is placed in this register. If the preceding instruction was a branch not taken, DEPC may be implemented point directly to the instruction in the delay slot. Execution of the DERET instruction causes a jump to the address in the DEPC. If the DEPC is both written from software (by MTC0) and by hardware (debug exception) then the DEPC is loaded by the value generated by the hardware. Bit 0 of the DEPC indicates the MIPS16 mode, and is 1 when the interrupted instruction is a MIPS16 instruction. Bit 0 always set to 0 for RC32334.	R/W	Undefined

Table 21.7 Debug Exception Program Counter

Notes

Debug Exception Save Register (DESAVE)

In the RC32334, this register is external to the core and implemented at physical address 0xFFFF-E210.

Bit(s)	Mnemonic	Description	R/W	Reset
31:0	DESAVE	This register is used by the debug exception handler to save one of the GPRs, that is then used to save the rest of the context to a pre-determined memory are, e.g. in the EJTAG Probe. This register allows the safe debugging of exception handlers and other types of code where the existence of a valid stack for context saving cannot be assumed.	R/W	Undefined

Table 21.8 Debug Exception Save Register

Register Map

The following registers are implemented in a Debug Support Unit. These registers contain the data, address, control and status of the break channels, and are only accessible for read when the processor is executing in Debug Mode (DM='1') and for write when DM=1 and the memory protection bit is switched off (MP='0'). When these conditions are not met, an attempt to access will cause an undefined result, e.g., invalid data may be read or a bus/address error exception may be raised. The DSU registers are non-cached memory locations, although they are in the kseg2 area. Only word/double word accesses are allowed to these registers. The base address for all of these registers is: 0xFF300000 and the actual address can be obtained by adding the offset value in Table 21.9.

Offset	Mnemonic	Description
0000	DCR	Debug Control Register
0004	IBS	Instruction Address Break Status
0008	DBS	Data Break Status
000C	PBS	Processor Break Status
0100	IBA0	Instruction Address Break 0
0104	IBC0	Instruction Address Break Control 0
0108	IBM0	Instruction Address Break Mask 0
0110	IBA1	Instruction Address Break 1
0114	IBC1	Instruction Break Control 1
0118	IBM1	Instruction Address Break Mask 1
0200	DBA0	Data Address Break 0
0204	DBC0	Data Break Control 0
0208	DBM0	Data Address Break Mask 0
0300	PBA0	Processor Address Bus Break 0
0304	PBD0	Processor Data Bus Break 0
0308	PBM0	Processor Data Bus Mask 0
030C	PBC0	Processor Bus Break Control 0 and Address Mask

Table 21.9 32-bit Register Map (Base Address = 0xff30 0000)

Debug Control Register

The Debug Control Register is located at address-offset 0x0000.

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
0	TM	Trace Mode 0: This mode will output PC trace information at the jtag_tdo/ejtag_tpc pin in real-time. The serial address output may be incomplete. 1: This mode will output the complete PC address as trace information at the jtag_tdo/ejtag_tpc pin. The real-time behavior of the processor is not guaranteed. The RC32334 implements a single level deep buffer to store PC trace information.	R/W	0
1	MRst*	Mask Soft Reset 0: Soft Reset to the core is masked during debug mode (DM='1') 1: No effect Soft Reset is always permitted during normal mode. This bit will not mask the Processor reset bit PrRst in EJTAG_Control_Register[16].	R/W	1
2	MP	**Memory Protection** 0: Write to the DSU + EJTAG reserved area (0xFF20-0000 - 0xFF3F-FFFF) is possible in debug mode. 1: Write to the DSU + EJTAG reserved area (0xFF20-0000 - 0xFF3F-FFFF) is protected in debug mode, except for the Debug Control Register (DCR). When the processor is not in debug mode, accesses to this area are NOT allowed.	R/W	1
3	MNmi	**Mask Non-Maskable Interrupt** (in non debug mode) 0: Mask the NMI signal to the core. 1: Enable the NMI signal to the core. In debug mode all interrupt inputs to the core are masked	R/W	1
4	MInt	**Mask Interrupt** (in non debug mode) 0: Mask the interrupt inputs [int(5:0)] to the core 1: Enable the interrupt inputs [int(5:0)] to the core In debug mode all interrupt inputs to the core are masked.	R/W	1
5		**Unused**	W1/R	0
6		**Unused**	W0/R	0
7.28	reserved	reserved	R	0
29	ENM	**Endianess** This bit indicates the default endianess. For some implementations it is a copy of the END bit in the core's CONFIG register. 0: Little Endian 1: Big Endian	R	0
30	HIS	**Halt Status** It indicates the sleeping state (power-down) when the debug exception was taken. The precise definition of this power down mode is implementation specific. 0: Processor was not in sleeping state 1: Processor was in sleeping state When the RC32334 executes the WAIT instruction, this bit is set.	R	0
31		**Unused**	R	0

Table 21.10 Debug Control Register - DCR

Notes

Instruction Address Match Registers

Instruction Address Break Status

Bit(s)	Mnemonic	Description	R/W	Reset
0	BS0	**Break Status 0** This bit, when set, indicates that an *instruction address break* or *instruction address trigger* has occurred. BS0 can be cleared by activating JtagRst, hard reset and also by writing a '0' to it.	R//W	0
14..1	BS[1] BS*n*	**Break Status [1]** These bits are similar to the BS0 bit and are implemented according to the number of channels available.	R//W	0
23..15	reserved	reserved	R	0
27..24	BCN	**Break Channel Number**: These bits indicate the total number of channels implemented for instruction address break. 0000: Reserved 0001: Channel 1 0002: Channel 2 (implemented value) ... 1111: Channel 15	R	0010
31..28	reserved	reserved	R	0

Table 21.11 Instruction Address Break Status Register - IBS

Instruction Address Break n

This register contains the upper 30/62 bits of the Instruction Address Break. Table 21.12 shows the format of the Instruction Address Break *n* register. This address is a virtual address.

Bit(s)	Mnemonic	Description	R/W	Reset
0	-	Zero	R	0
1 1	reserved	reserved	R	0
31..2	IBA*n*[31..2]	Instruction Address Break *n*	R/W	?

Table 21.12 Instruction Address Break Register n - IBAn

Instruction Address Break Mask n

These registers specifies the mask value for the Instruction Address Break Register *n* (IBA*n*). Each bit corresponds to a bit in the address register, and when:

- *0: Address bit is not masked, address bit is compared.*
- *1: Address bit is masked, address bit is not compared.*

Bit(s)	Mnemonic	Description	R/W	Reset
0		Zero	R	0
1 1	reserved	reserved	R	0
31..2	IBM*n*[31..2]	Instruction Address Break Mask n	R/W	?

Table 21.13 Instruction Address Break Mask Register n - IBMn

Notes

Instruction Address Break Control n

This register selects the instruction address match function to enable debug break or trace trigger.

Bit(s)	Mnemonic	Description	R/W	Reset
0	BE	**Break Enable** This bit enables the Instruction Address break function. 0: Instruction Address break function is disabled 1: Instruction Address break function is enabled If the Instruction Address break function is valid and the processor's virtual Instruction Address and the address set by the IBA*n* register (masked by IBM*n*) match, a debug exception to the processor is generated. The BS*n* bit in the Instruction Address Break Status register is set and the DIB bit in the Debug Register is set to identify the cause of the debug exception. When the Instruction Address break occurs, the debug exception happens just *before* the instruction is executed. If the debug exception handler is already running (DM='1'), then the debug exception will not be taken.	R/W	0
1	reserved	reserved	R	0
2	TE	**Trace Trigger Enable** This bit enables the Trace Trigger function. 0: Instruction Address trace trigger function is disabled 1: Instruction Address trace trigger function is enabled. If the Trace Trigger function is valid and the processor's virtual Instruction Address and the address set by the IBA*n* register (masked by IBM*n*) match, the trace trigger information TST(010) or TSQ(001) is output to the ejtag_pcst[2:0] pins; also the BS0 bit in the Instruction Address Break Status register bit is set. When an address match occurs with both BE='1' and TE='1', the Instruction Address break exception is taken *after* the trace trigger information is output to the ejtag_pcst[2:0] pins.	R/W	0
31..3	reserved	reserved	R	0

Table 21.14 Instruction Address Break Control n Register - IBCn

Data Address and Data Match registers

Data Address Break Status

This register provides the status of the possible 15 Data Breakpoints.

Bit(s)	Mnemonic	Description	R/W	Reset
0	BS0	**Break Status 0** This bit, when set, indicates that a *data address break* or *data address trigger* has occurred. BS0 can be cleared by activating JtagRst, hard reset and also by writing a '0' to it.	R/W	0
14..1	BS[1] BS*n*	**Break Status [1]** These bits are similar to the BS0 bit and are implemented according to the number of channels available.	R/W	0
23..15	reserved	reserved	R	0
27..24	BCN	**Break Channel Number** These bits indicate the total number of channels implemented for data address break. 0000: Reserved 0001: Channel 1 (implemented value) ..... 1111: Channel 15	R	0001
31..28	reserved	reserved	R	0

Table 21.15 Data Address Break Status - DBS

Notes

Data Address Break n

This register contains the upper 30 bits of the Data Address Break DBA*n*. This address is a virtual address.

Bit(s)	Mnemonic	Description	R/W	Reset
0	W	Data break match on write 0: Data Address break disable for writes 1: Data Address break enabled for writes	R	0
1	R	Data break match on reads 0: Data Address break disable for reads 1: Data Address break enabled for reads	R	0
31..2	DBA*n*[31..2]	Data Address Break *n*	R/W	?

Table 21.16 Data Address Break n Register - DBAn

Processor Bus Match Registers

The Processor Bus Match registers monitor the bus interface of the MIPS CPU and provide debug exception or trace trigger for a given physical address and data. Since the CPU bus is implementation specific, Processor Bus Breaks may not work identically for different MIPS CPUs.

Processor Bus Break Status

The following table shows the format of the Processor Bus Break Status register.

Bit(s)	Mnemonic	Description	R/W	Reset
0	BS0	**Break Status 0** This bit, when set, indicates that a *processor bus break* or *processor bus trigger* has occurred. BS0 can be cleared by activating JtagRst, hard reset and also by writing a '0' to it.	R/W	0
14..1	BS[14..1] BS*n*	**Break Status [14..1]** These bits are similar to the BS0 bit and are implemented according to the number of channels available.	R/W	0
23..15	reserved	reserved	R	0
27..24	BCN	**Processor Bus Break Channel Number** These bits indicate the total number of channels implemented for Processor Bus Break. 0000: Reserved 0001: Channel 1 (implemented value) ..... 1111: Channel 15	R	0001
31..28	reserved	reserved	R	0

Table 21.17 Processor Bus Break Status - PBS

Processor Address Bus Break n

This register contains the bits of the physical Processor Address Bus Break.

Bit(s)	Mnemonic	Description	R/W	Reset
0	reserved	reserved	R	0
1 1	reserved	reserved	R R/W	0 ?
31..2	PBA*n*[31..2]	Processor Address Bus Break *n*	R/W	?

Table 21.18 Processor Address Bus Break Register n - PBAn

Notes

Processor Data Bus Break n

This register specifies the data value for the Processor Data Bus match.

Bit(s)	Mnemonic	Description	R/W	Reset
31..0	PBD*n*[31..0]	Processor Data Bus Break *n*	R/W	?

Table 21.19 Processor Data Bus Break n Register - PBDn

Processor Data Bus Mask n

This register specifies the mask value for the Processor Data Bus Break register. Each bit corresponds to a bit in the data register:

- *0: Data bit is not masked, data bit is compared*
- *1: Data bit is masked, data bit is not compared*

Bit(s)	Mnemonic	Description	R/W	Reset
31..0	PBM*n*[31..0]	Processor Data Bus Mask *n*	R/W	?

Table 21.20 Processor Data Bus Mask n Register - PBMn

Processor Bus Break Control and Address Mask n

This register selects the Processor Bus match function to enable debug break or trace trigger. It also includes control bits to enable comparison as well as mask bits to exclude address bits from comparison.

Bit(s)	Mnemonic	Description	R/W	Reset
0	BE	**Break Enable** This bit enables the Processor Bus break function. 0: Processor Bus break function is disabled 1: Processor Bus break function is enabled If the Processor Bus break function is valid and the Processor's physical Address = PBA*n* register (masked by LAM) and the Processor's Data bus = PBD*n* register (masked by PBM*n*), then a debug exception to the processor is generated. The BS*n* bit in the Processor Bus Break Status register is set and the DINT bit in the Debug Register is set to identify the cause of the debug exception. If the debug exception handler is already running (DM='1'), then the debug exception will not be taken.	R/W	0
1	reserved	reserved	R	0
2	TE	**Trace Trigger Enable** This bit enables the Trace Trigger function. 0: Processor Bus trace trigger function is disabled 1: Processor Bus trace trigger function is enabled. If the Trace Trigger function is valid and the Processor's physical Address = PBA*n* register (masked by LAM) and the Processor's Data bus = PBD*n* register (masked by PBM0), then the trace trigger information TST(010) or TSQ(001) is output to the ejtag_pcst[2:0] pins; also the BS*n* bit in the processor Bus Break Status register is set. When a processor bus match occurs with both BE='1' and TE='1', the Processor Bus break exception is taken *after* the trace trigger information is output to the ejtag_pcst[2:0] pins.	R/W	0
3	reserved	reserved	R	0
4	IFUC	Instruction Fetch From Un-cached Area This bit enables the comparison on Processor Address and Data Bus for Instruction Fetches in the un-cached area. 0: Processor Address and Data Bus is not compared for Instruction Fetches in the un-cached area. 1: Processor Address and Data Bus is compared for Instruction Fetches in the un-cached area. When BE='1' and IFUC='1' the debug break exception is taken on the same instruction.	R/W	0

Table 21.21 Processor Bus Break Control and Address Mask n - PBCn (Part 1 of 2)

Notes

Bit(s)	Mnemonic	Description	R/W	Reset
5	DLUC	Data Load from un-cached Area This bit enables the comparison on Processor Address and Data Bus for Data Loads in the un-cached area. 0: Processor Address and Data is not compared for Data Load in the un-cached area. 1: Processor Address and Data is compared for Data Load in the un-cached area. When BE='1' and DLUC='1' the debug break exception is taken *after* the next instruction.	R/W	0
6	DSUC	**Data Store to un-cached Area** This bit enables the comparison on Processor Address and Data Bus for Data Store to the un-cached area. 0: Processor Address and Data is not compared for storing data into the un-cached area. 1: Processor Address and Data is compared for storing data into the un-cached area. When BE='1' and DSUC='1' the debug break exception is taken *after* the next instruction.	R/W	0
7	DSCA	**Data Store to Cached Area** This bit enables the comparison on Processor Address and Data Bus for Data Store to the Cached area. 0: Processor Address and Data is not compared for storing data to the Cached area. 1: Processor Address and Data is compared for storing data to the Cached area. When BE='1' and DSCA='1' the break exception is taken *after* the next instruction.	R/W	0
31..8	LAM	Lower Address Mask These bits specify the mask value for the 24 bit lower bits of the Processor Address Bus Break register (PBA*n*[23..0]). Each bit corresponds to the same bit in PBA*n*. 0: Address bit is not masked, address bit is compared. 1: Address bit is masked, address bit is not compared.	R/W	0x000

Table 21.21 Processor Bus Break Control and Address Mask n - PBCn (Part 2 of 2)

Processor Bus Break Function

Processor bus break becomes effective by setting Processor Bus Control Register bits. The Debug Unit will monitor the processor bus and, depending on the bit setting for instruction fetch from Uncache area or data load/store in Uncache or Cache region (i.e., IFUC, DLUC, DSUX, PBCO bits), address and data comparison is performed. PBAO, PBDO, and PBM are holding the address, data, and mask value to be compared for debug interruption.

Processor Bus Trace Trigger Function

By setting TE=1 bit in the Processor Control register, the processor bus trace trigger becomes effective. The Debug Unit will monitor the processor bus and, depending on the bit setting for instruction fetch from Uncache area or Data load/store in Uncache or Cache region (i.e., IFUC, DLUC, DSUC, PBCO bits), address and data comparison is performed. When the address set by PBA0register and the data set by PBD0 register matches according to data mask value, Trace Information TST(010) or TSQ(001) is output to PCST[2:0].

Notes

Debug Exception

The debug exception has priority over all exceptions, except the reset exception.

Debug Exception Causes

There are several causes of the Debug Exception:

- *Software Debug Breakpoint (SDBBP) instruction execution*
- *Match on Hardware DSU registers*
- *Debug Exception from the JTAG port. This is caused by the EJTAG Probe setting the Jtagbrk bit in the EJTAG_Control_Register.*

During debug mode no other debug exception can be taken.

Debug Exception Enabling/Disabling

The causes of the Debug Exception can be masked as follows:

- *The Software Debug Breakpoint (SDBBP) instruction execution is masked in debug mode.*
- *The Match on Hardware DSU registers is enabled by setting the BE bit in the corresponding Control register.*
- *Debug Exceptions from the JTAG port are only masked in debug mode.*

Debug Exception Handling

When the debug exception is raised, the processor jumps to the debug exception handler.

- *If the ProbEn bit in the EJTAG_Control_Register[15] is set, the debug exception vector is located at address location: 0xFF20-0200. (This is mapped in un-cacheable address space).*
- *If the ProbEn bit in the EJTAG_Control_Register[15] is cleared, the debug exception vector is located at address location: 0xBFC0-0480. (This is mapped in un-cacheable address space).*
- *Only the contents of the Debug register and the DEPC will be affected by the debug exception.*
- *The Debug Mode bit (DM) in the Debug register is set to '1'.*
- *One (or more) of the following bits in the Debug Register are set to identify the cause of the debug exception:*
  - *DSS: after single step execution of an instruction and the SSt bit in the Debug register is set.*
  - *DBp: after execution of the SDBBP instruction.*
  - *DDBL: Data Address match during a Load memory instruction.*
  - *DDBS: Data Address match during a Store memory instruction.*
  - *DIB: Instruction Address match.*
  - *DINT: Processor Bus match or JtagBrk.*
  - *DBD: Set to '1' when the exception was raised for an instruction in the branch delay slot.*
  - *NIS: Set to '1' if a non-maskable interrupt occurred at the same time as the debug exception.*
  - *UMS: Set to '1' when the TLB exception occurred at the same time as the debug exception.*
  - *OES: Set to '1' if another exception (other than reset, TLB, NMI) was raised at the same time as the debug exception.*

Exception priorities: DIB have a higher priority than DBp, and Jtagbrk has the lowest priority.

In case of SDBBP caused exception:

- *The DEPC register points to the SDBBP instruction, unless that instruction is in the branch delay slot, in which case the DEPC register points to the branch instruction and DBD bit is set to '1'.*

In case the debug exception had other cause besides SDDBP:

- *The DEPC register points to the address of the instruction where the exception was raised (for single step exception, this is the instruction to be executed).*
- *A single step exception is not raised for an instruction in the branch delay slot.*
- *When the DERET instruction is executed, a single step exception is not raised for an instruction at the return destination. If the return destination is a branch instruction, a single step exception is not raised for that branch instruction or for the instruction in the branch delay slot.*

Notes

Exception Handling when in Debug Mode (DM bit is set)

In Debug Mode, the processor core can only take reset type exceptions, all other exceptions are not taken. All interrupts including NMI are masked. When the NMI interrupt occurred during Debug mode it is stored internally and the NMI interrupt is taken after debug handler is finished (DM = '0').

A Load or Store Instruction which generated a TLB related exception during Debug Mode is not taken and is not executed. Only the TLF bit in Debug Register[11] will be set.

When a Load or Store instruction causes a bus error exception when the processor is in Debug Mode, no exception is taken and the BsF bit in the Debug Register is set. The result of Load/Store operation is discarded.

The debug mode has the same privileges as the kernel mode, i.e. access to all physical memory, the complete instruction set and all registers including GPR and Coprocessor 0 instructions, regardless of the value of the Kuc bit.

Servicing the Debug Exception

When a debug exception occurs, the debug exception handler should save the context of the program that was executing. For that, it can use the DESAVE register. After that, the service routine should determine the nature of the exception from the Debug Register bits and invoke the corresponding exception handler.

The DEPC register holds the address to where processing resumes after the debug exception routine has finished. The address that has been loaded in the DEPC register is the virtual address of the instruction that caused the debug exception. If the instruction is in the branch delay slot, the virtual address of the immediately preceding branch or jump instruction is placed in this register and the DBD bit is set. Execution of the DERET instruction causes a jump to the address in the DEPC.

In case of SDBBP caused exception: the unused bits of the SDBBP instruction (indicated as CODE) can be used for passing additional information to the exception handler. In order to allow these bits to be viewed at, the user program should load the contents of the memory word containing this instruction, using the DEPC register. When the DBD bit in the Debug register is set to '1', the SDBBP instruction is in the branch delay slot, therefore the value in the DEPC register should be added with 4.

PC Trace

The basic idea of the instruction trace method is to output the virtual address of an instruction only when the program flow is changed by a jump instruction or exception. Jump instructions can be divided into the following two groups:

- ***PC Relative Jump and Direct Jump****: the target address of these instructions is fixed and identified by the source program. The target address is usually specified by a "label" in assembly language e.g. j label1 (jump to label1).*
- ***Indirect Jump****: the indirect jump instruction jumps to an address contained in a general register. This instruction is usually used for a subroutine call or table jump. The target address is determined during program execution, e.g. jr r1 (jump to contents of register r1). Note that the ERET instruction is treated as an indirect jump too.*

A target address of a PC relative or direct jump instruction can be determined by the instruction itself. However, a target address of an indirect jump depends on the contents of a register when the instruction is executed. Therefore the processor should output a target address of an indirect jump for real-time trace information.

Jump instructions are also classified into conditional and unconditional jumps. The dynamic information whether the conditional branch is taken or not taken is necessary for instruction trace.

Notes

Jumps	Conditional	Unconditional
PC Relative instruction Direct Jump instruction	Taken / Not Taken	-
Indirect Jump instruction	Target Address Taken / Not Taken	Target Address

Table 21.22 Dynamic Trace Information

Instruction Trace Method

The EJTAG module requires output pin(s) for the PC trace information. EJTAG uses at least the data output jtag_tdo/ejtag_tpc for that. More pins can be dedicated for PC output if the Extended EJTAG interface is used (see PC Status and Exception Vector Encoding section).

The other signals (ejtag_pcst) show the *status of execution* and also show when one of the break channels (when programmed to output a trigger) has found a match.

In the RC32334, the number of ejtag_tpc bits output is 30. To reduce the information at the ejtag_tpc pin(s), the processor only outputs a target address of a Direct Jump, an Indirect Jump, a Branch instruction and (part of) exception vector addresses. However, there is the possibility that the target address output is not complete.

The target address of an indirect jump may take 30 cycles to output the target address at the 1 bit jtag_tdo/ejtag_tpc pin. If the next indirect jump is executed in 30 cycles, then the first target address is not output completely.

In PC Trace mode, non-sequential Program Counter Address information (PC Trace) is output at the ejtag_tpc pin(s), in conjunction with trace information at the ejtag_pcst pins. Non-sequential PC trace is output when there is a change in the program flow, caused by:

- *Direct Jump Instructions (J and JAL) where the target address is defined.*
- *Indirect Jump Instructions (JR, JALR and ERET) where the target address is contained in a register.*
- *Branch Instructions (BEQ, BNE, BLEZ, BGTZ, BGEZ, BLTZ, BLTZAL, BCzT, BCzF, BEQL, BNEL, BLEZL, BGTZL, BLTZL, BGEZLL, BGEZAL, BLTZALL, BCzTL and BCzFL) where a branch target address is computed from the sum of the address of the instruction in the delay slot and the 16-bit offset.*
- *Interrupts and exceptions: an exception code is then output at the ejtag_tpc pin(s).*

PC Status and Exception Vector Encoding

PC Status Encoding

The PC Trace Status (ejtag_pcst) Information is output at the same rate as the CPU pipeline clock. The PC status is only active in Real-Time mode. The ejtag_pcst encodes the status of the MIPS CPU execution as follows.

PCST	Symbol	Function
1 1 1	STL	Pipeline stall. During this state there is no Trace Trigger output.
1 1 0	JMP	Execution of a Taken Jump Instruction. This status indicates that the jump instruction is taken and also indicates the start of the target PC address output. In this case the target PC address of this jump will be output.
1 0 1	BRT	Execution of a Taken Direct Jump Instruction or PC Relative instruction. This status indicates the direct or PC relative jump is taken. In this case there is *no* PC trace output of this jump's target address.
1 0 0	EXP	Exception generated. This status indicates that an exception occurred, and an exception code is output at the ejtag_tpc pin(s).

Table 21.23 PC Trace Status Information (Part 1 of 2)

Notes

PCST	Symbol	Function
0 1 1	SEQ	Execution of non Jump instructions. This status information indicates that the processor has executed one instruction of sequential (in line) code. This status also indicates that the conditional jump is not taken.
0 1 0	TST	Trace Trigger information is output when the pipeline is stalled. This condition shows that an Address, Data or Processor Bus trace trigger has occurred before the time that the pipeline is stalled.
0 0 1	TSQ	Trace Trigger output at execution time. This condition shows that an Address, Data or Processor Bus trace trigger has occurred during processor execution.
0 0 0	DBM	Debug Mode. This condition is active when the Debug Mode is on (DM = '1'). This code *may* also be output when trace in not on and the CPU is in Normal Mode.

Table 21.23 PC Trace Status Information (Part 2 of 2)

Status Output on Delay Slots

All jump and branch instructions have a delay slot. The instruction in the delay slot is normally executed prior to the jump/branch target instruction, however, some instructions nullifies (kills) the delay slot instruction rather than executing it. These instructions are:

- *Branch likely not taken instructions.*
- *The ERET instruction.*

For the nullified delay slot instructions the STL (or TST) code is output since the instruction is not part of the actual instruction flow; for executed delay slot instructions the SEQ (or TSQ) code is output.

For the jump/branch instruction itself JMP/BRT is output when the jump/branch is taken, and SEQ (or TSQ) is output for the branch when it is not taken. JMP is always output for the ERET instruction. For a branch likely not taken instruction SEQ is output for the branch likely and STL is output for its nullified delay slot.

Note that the PC Trace interpreting software may not be able to determine the exact target of a jump/branch/ERET instruction unless the source is known; this is true even if a complete PC is output for the target. The reason for this, is that an instruction resulting in a JMP code may or may not have an executed delay slot (only known if source is known), and thus it will either be the first or the second significant code (code other than STL or TST) after the JMP which will represent the instruction at the target PC. The PC Trace interpreting software will however in most cases be able resolve this uncertainty when the first JMP or BRT is met in the program code at the target PC.

Exception Vector Encoding

When an instruction receives an exceptional event, either due to an external source (e.g. interrupt) or as part of the execution flow (SYSCALL, overflow etc.), the EXP code is output for that instruction instead of what would otherwise have been output.

During an exception, when ejtag_pcst shows the EXP code, the ejtag_tpc pins output a exception vector code, starting from the LSB of the code. Instructions that generate a Debug Exception will not output the EXP code nor the exception code at the ejtag_tpc pin(s).

Exception Vector Encoding for RC32334:

Table 21.24 shows the 4 bit exception code output at the ejtag_tpc pin(s) during the EXP code at the ejtag_pcst pins.

Exception	BEV	EXL	A[29]	A[9]	A[8]	A[7]
Reset, Softreset, NMI	-	-	1	0	0	0
TLB Refill	0	0	0	0	0	0

Table 21.24 Exception and Exception Codes at ejtag_tpc (Part 1 of 2)

Notes

Exception	BEV	EXL	A[29]	A[9]	A[8]	A[7]
Cache Error	0	-	0	0	1	0
Other	0	-	0	0	1	1
Interrupt (Cause.IV=1)	0	0	0	1	0	1
TLB Refill	1	0	1	1	0	0
Cache Error	1	-	1	1	1	0
Interrupt (Cause.IV=1)	1	0	1	0	0	1
Other	1	-	1	1	1	1

Table 21.24 Exception and Exception Codes at ejtag_tpc (Part 2 of 2)

External Interface Definition

EJTAG

The following signals are used during the PC Trace mode (Table 21.25 shows the complete list of EJTAG interface signals).

- ***jtag_tdo/ejtag_tpc****: During PC Trace Mode, the jtag_tdo/ejtag_tpc provides a non-sequential PC (ejtag_tpc) at the processor clock. ejtag_tpc is output simultaneously with the Program Counter Trace Status Signals ejtag_pcst, starting with PC address 2 or 1. depending on the support of MIPS16 or not.*
- ***ejtag_pcst[2:0]:*** *PC Status Trace Information, with the encoding described in Table 21.23.*
- ***ejtag_dclk:*** *Processor Clock: This signal is used by the external EJTAG Probe to capture the ejtag_tpc and ejtag_pcst signals at the MIPS CPU clock rate. The ejtag_tpc and ejtag_pcst signals are output at the positive edge of ejtag_dclk.*

Priority of Target Address Output (ejtag_tpc)

The target address output at jtag_tdo/ejtag_tpc may change due to occurrence of an exception or of a next Jump or Branch instruction. There are priorities specified at which the ejtag_tpc output will change. The Trace Mode (TM) bit in the Debug Control Register (DCR[0]) determines if the current target PC output is stopped and the new target PC started instead, or that the current target PC is completely finished.

Real Time ejtag_tpc Output (TM='0' in DCR[0])

During real-time ejtag_tpc output, the PC trace information is output at the processor clock and the PC trace information is in sync. with the program execution. The target PC address output may be incomplete. The priorities for target PC output in this mode are:

1. If there is no ejtag_tpc being output, the target address of a taken jump will be output at jtag_tdo/ejtag_tpc, also when it is a Direct Jump. The ejtag_pcst pins will show the JMP code (see Figure 21.13).
2. If a new indirect jump is executed while the previous target PC is being output, the new indirect jump target PC will always start and the previous target PC output will be aborted (see Figure 21.14).
3. If an exception occurs while the previous target PC is being output, an exception vector code is output and then the previous discontinued PC output is resumed (see Figure 21.16).
4. If an exception occurs while a previous exception vector code is being output, the previous exception vector code output is aborted and the new exception vector code output is started.
5. If a new direct jump or branch is executed while the previous target PC is being output, then this new direct jump or branch target PC will not be output. Instead the ejtag_pcst code will indicate the BRT code. The target PC for the direct jump or branch is only output when there is no PC trace output for another jump/branch going on (see Figure 21.13). If an exception vector code output is gong on but no jump/branch target PC is pending, then JMP is output for the direct jump and the target PC output for the direct jump starts once the exception vector code has been output.
6. If a jump occurs after exception, ejtag_tpc outputs exception code first and then the target address.

Notes

Non-Real Time ejtag_tpc Output (TM='1' in DCR[0])

During non-real time trace mode the 30 bit target PC for indirect jumps and the 3 bit exception vector code is always output completely. In this mode, it is only guaranteed that all indirect jump target addresses and exception vector codes are fully output on ejtag_tpc, this is however enough information to completely reconstruct the program execution flow. The RC32334 implements a single level deep buffer to store the PC trace address.

The priorities for the PC trace output are:

1. If there is no ejtag_tpc being output, the target address of a taken jump will be output at jtag_tdo/ ejtag_tpc, also when it is a Direct Jump. The ejtag_pcst pins will show the JMP code.
2. If an exception occurs while the target PC is being output, the exception vector code is output first and then the previous discontinued PC output is resumed. The Processor core is not stalled in this case.
3. If an exception occurs while a previous exception vector code is being output, the pending exception vector code is output first and then the new exception vector code is output. The Processor core is stalled in this case.
4. ***a.*** If an indirect jump instruction is executed while the previous target PC is being output, then the Processor Core is stalled until the previous target PC is completely output.
   b. If an indirect jump instruction is executed while the previous target PC from a direct jump is being output, the RC32334 uses the 1 level deep buffer to store the PC trace address.
5. If a new direct jump or branch is executed while the previous target PC is being output, then this new direct jump or branch target PC will not be output. Instead the ejtag_pcst code will indicate the BRT code. The target PC for the direct jump or branch is only output when there is no PC trace output going on (see Figure 21.13).
6. If a jump occurs after exception, ejtag_tpc outputs exception code first and then the target address.

Examples of PC Trace Output

Conditional PC Relative Jump Instruction

Figure 21.13 indicates the execution of conditional PC relative instructions. The beq and bne instructions are conditional PC relative jumps. Because the first jump instruction (beq) is taken and the ejtag_tpc output is not in use, the target PC of the beq starts to output and the ejtag_pcst status is the 'JMP'. The jump status corresponding to the second jump (bne) is the 'SEQ' which indicates the jump is not taken. The third jump (bne) is taken, the ejtag_pcst lines show 'BRT' but there is no ejtag_tpc output from its target address since it is a Direct Jump and the ejtag_tpc line is already outputting.

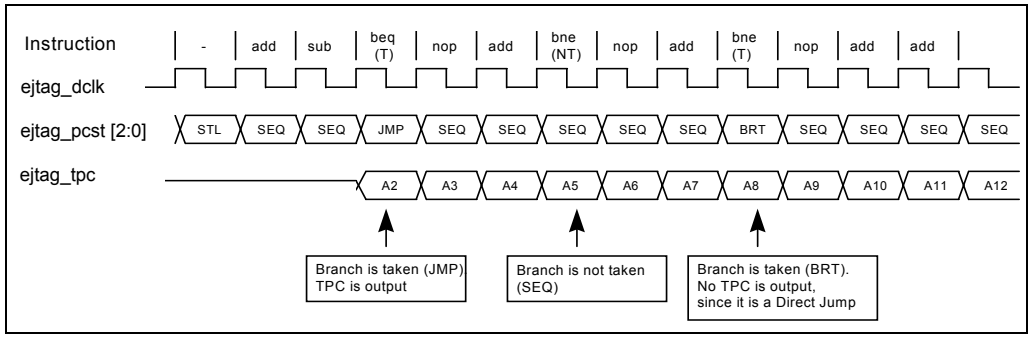


Figure 21.13 Trace of Conditional PC Relative Jump Instruction

Indirect Jump Instruction

The execution of an indirect instruction is shown in Figure 21.14. When the first indirect jump (jr1) instruction is executed, the processor outputs the 'JMP' code at the ejtag_pcst pins and starts to output its target address from the lower bit at the ejtag_tpc pin. The lower bit is A2. When the second indirect jump instruction (jr2) is executed, the processor stops outputting the target address of the first indirect jump and starts outputting the second target address. In this case, the target address of the first indirect jump is incomplete.

Notes

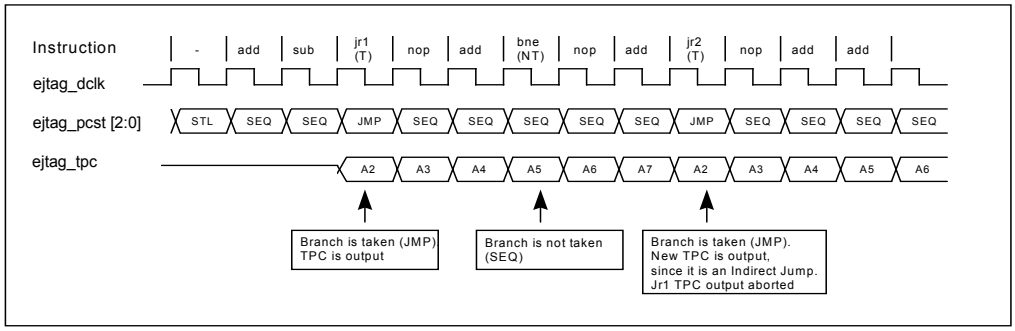


Figure 21.14 Trace of Indirect Jump Instruction

PC Trace Of An Exception Followed By A Jump Indirect Instruction

In Figure 21.15, the Break instruction is executed and causes an exception. This is indicated by the 'EXP' code at ejtag_pcst and the ejtag_tpc starts outputting the 3-bit exception code '001' starting with the LSB. The taken Jr2 instruction causes the JMP code at ejtag_pcst and the outputting of its target address at ejtag_tpc.

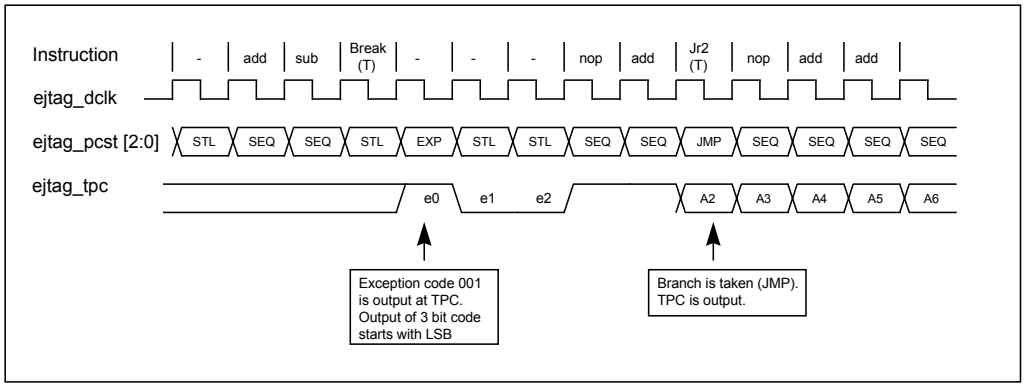


Figure 21.15 Trace of an Exception Followed by a Jump Indirect Instruction

PC Trace of an Indirect Instruction Followed by an Exception

In Figure 21.16, the indirect jump Jr1 starts the ejtag_tpc output, but the target address output is stopped to allow exception code bits of the exception to be output. After this the target address output is continued again.

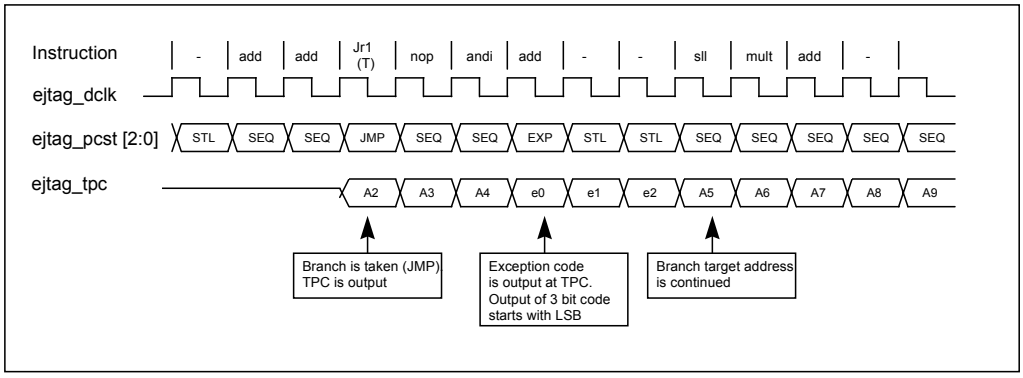


Figure 21.16 Trace of Indirect Jump Instruction Followed by an Exception

Notes

Examples of Trace Trigger Output

Trace trigger information is output at the ejtag_pcst pins when an instruction address, data or processor bus trigger occurred.

In general trace triggers should be indicated on the instruction which caused the trigger. However, since trigger indications can only be indicated in the PC Trace output on SEQ or STL codes (by replacing these codes with TSQ or TST) the trace trigger indication cannot be exactly defined. If JMP, BRT or EXP needs to be output, a simultaneous trigger indication must be output on another code and thus the EJTAG Probe cannot accurately determine the instruction that generated the trigger.

Instruction Address Trace Trigger

Figure 21.17 shows the occurrence of the Trace Trigger TSQ code at the ejtag_pcst pins for the instruction address that matches the required conditions.

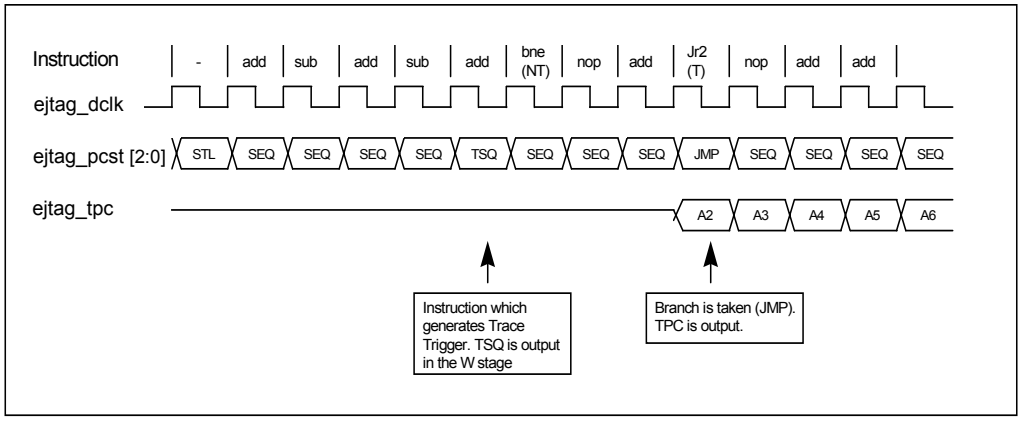


Figure 21.17 instruction Address Trace Trigger

Trace Trigger and General Exception at the Same Time

In Figure 21.18, both the Trace Trigger and an exception occur at the same moment, then the ejtag_pcst pins show the TST code, followed by the EXP code. The 3 bit exception code is output at ejtag_tpc.

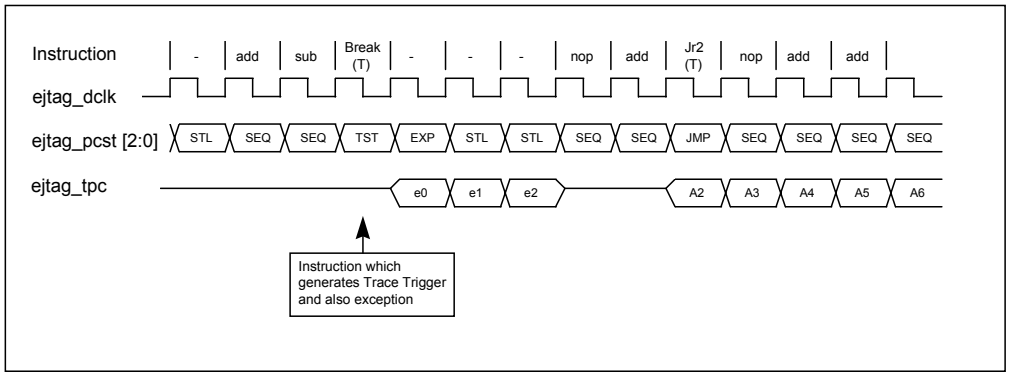


Figure 21.18 Trace Trigger and General Exception at the Same Time

Jump Indirect Causes Trace Trigger

In Figure 21.19 the Jump Indirect (Jr2) is the instruction that generates the Trace Trigger. This indicated by the TSQ code at the ejtag_pcst pins.

Notes

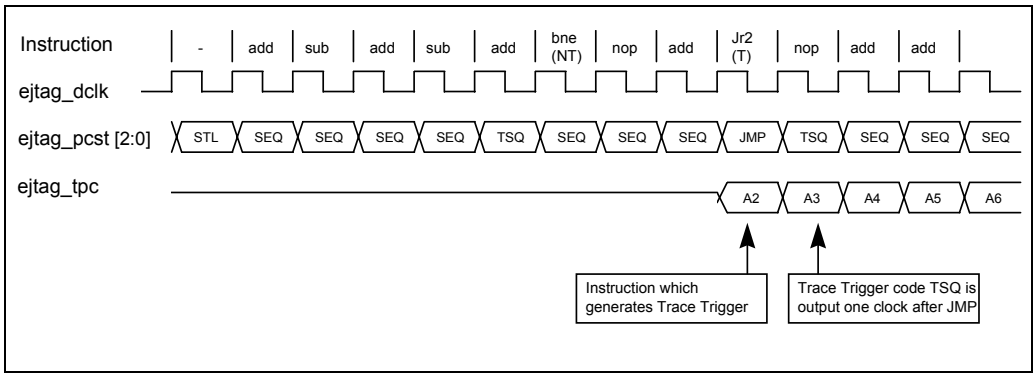


Figure 21.19 Jump Indirect Causes Trace Trigger

Instruction after Jump Indirect Causes Trace Trigger

In Figure 21.20 the Trace Trigger is caused by the instruction following the Jr2. The resulting trace trigger output information however is the same. The EJTAG Probe can not accurately determine the instruction that generated the trigger.

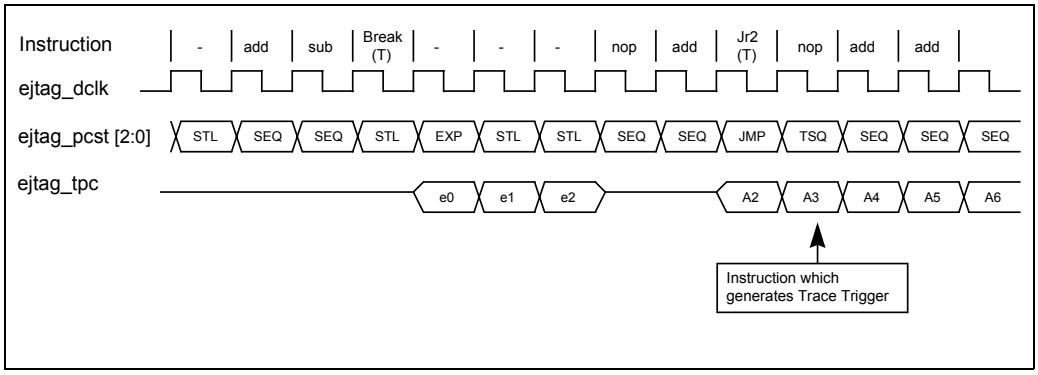


Figure 21.20 Instruction after Jump Indirect Causes Trace Trigger

Switching from Real-Time Trace to Debug

Real-Time Trace Mode to Debug Mode (No ejtag_tpc Output)

In Figure 21.21, the debug exception occurs in the instruction following the NOP instruction. In this case there is no target PC output going on. The debug mode is entered directly after the debug exception. When the instruction causing the debug exception is also set up for generating Trace Trigger, then the TST code is output at ejtag_pcst just before debug mode is entered.

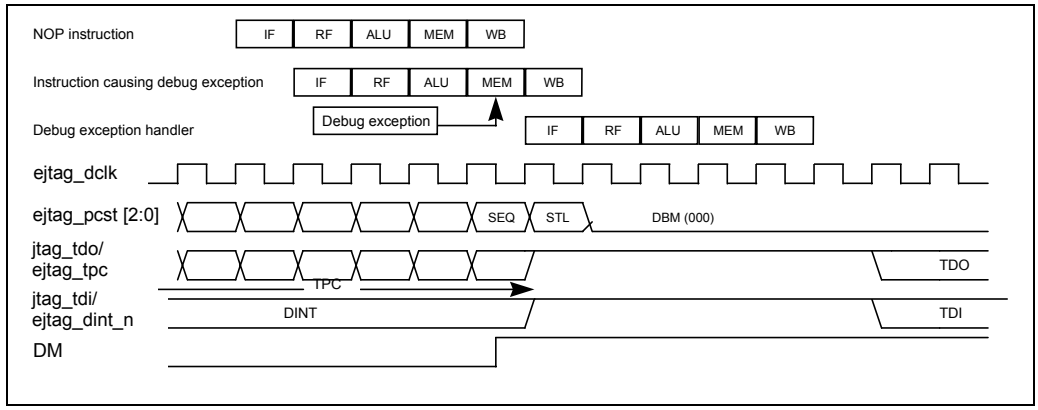


Figure 21.21 Real-Time Trace Mode to Debug Mode (No Tpc Output)

Notes

Real-Time Trace Mode to Debug Mode

In Figure 21.22, the target PC is being output (e.g. due to execution of an indirect JR instruction) when a debug exception occurs. In this case the Debug Mode is entered *after* the trace output is finished. During this time the STL code is output at ejtag_pcst; the debug mode entry is indicated by the DBM code. In debug mode, the jtag_tdo/ejtag_tpc pin function changes from ejtag_tpc to jtag_tdo; jtag_tdi/ejtag_dint_n pin function changes from ejtag_dint_n to jtag_tdi.

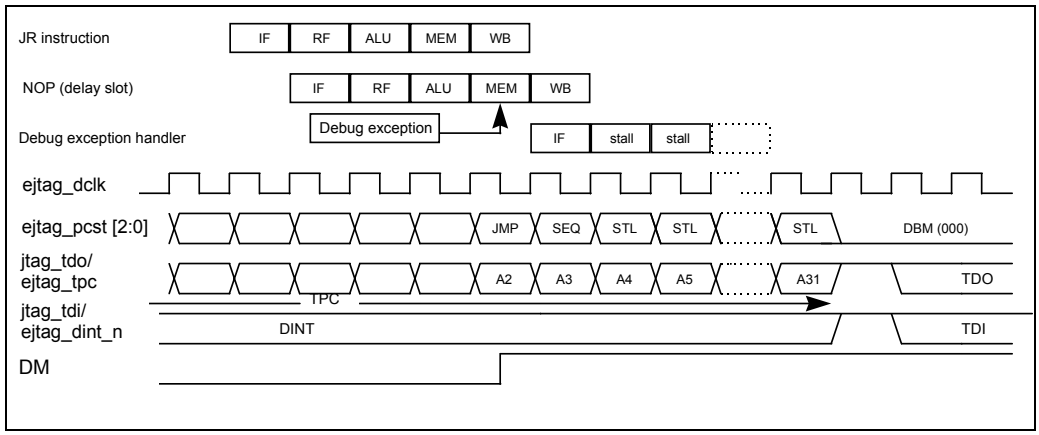


Figure 21.22 Real Time Trace Mode to Debug Mode (Debug Exception in Branch Delay Slot)

Pin Out of the Standard EJTAG

Figure 21.23 represents the timing diagram for the EJTAG interface signals.

The standard EJTAG connector (without PC Trace signals) is a 12-pin connector. For Standard EJTAG, a 24-pin connector has been chosen, providing 12 signal pins and 12 ground pins. This guarantees elimination of noise problems by incorporating signal-ground type arrangement.

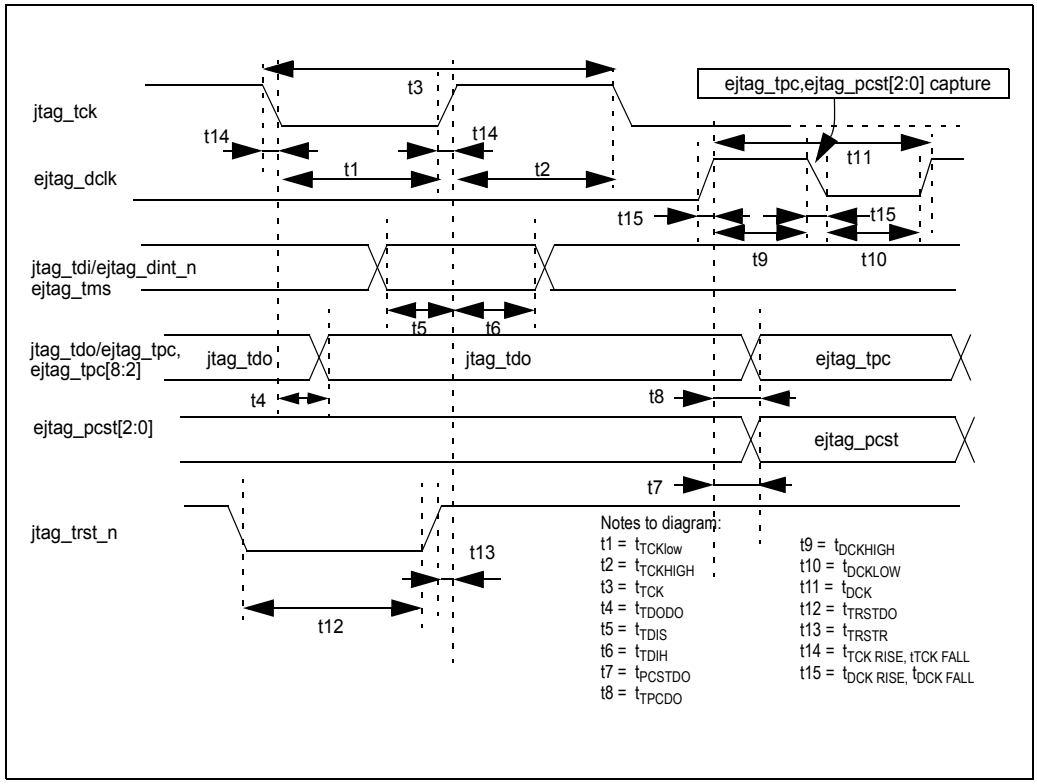


Figure 21.23 Timing Diagram of the EJTAG Interface Signals

Notes

Table 21.25 shows the pin numbering for the Standard EJTAG (EJT) connector. All the even numbered pins are connected to GROUND. The last columns show the target signal direction and the recommended termination at the target. Target termination resistors may be internally in the chip or externally on the board.

PIN	SIGNAL	TARGET I/O	TERMINATION[1]
1	ejtag_trst_n (optional)	Input	10 kΩ pull-down resistor
3	jtag_tdi, ejtag_dint_n	Input	10 kΩ pull-up resistor
5	jtag_tdo/ejtag_tpc	Output	33 Ω series resistor
7	jtag_tms	Input	10 kΩ pull-up resistor
9	jtag_tck	Input	10 kΩ pull-up resistor[2]
11	system reset	Input	10 kΩ pull-up resistor
13	ejtag_pcst[0]	Output	33 Ω series resistor
15	ejtag_pcst[1]	Output	33 Ω series resistor
17	ejtag_pcst[2]	Output	33 Ω series resistor
19	ejtag_dclk	Output	33 Ω series resistor
21	ejtag_debugboot	Input	10 kΩ pull-down resistor
23	VIO	Input	Must be connected to the VCC I/O supply of the device.

Table 21.25 Pin Numbering of the JTAG and EJTAG Target Connector

1. The value of the series resistor may depend on the actual PCB layout situation.

2. jtag_tck pull-up resistor is not required according to the JTAG (IEEE1149) standard. It is indicated here to prevent a floating CMOS input when the EJTAG connector is unconnected.

EJTAG Application Information

Using JTAG Boundary Scan and EJTAG

Figure 21.24 gives an application diagram of a target board showing how the processor's EJTAG signals are connected to the Target Connector and to the other (boundary Scan) IC's on the board.

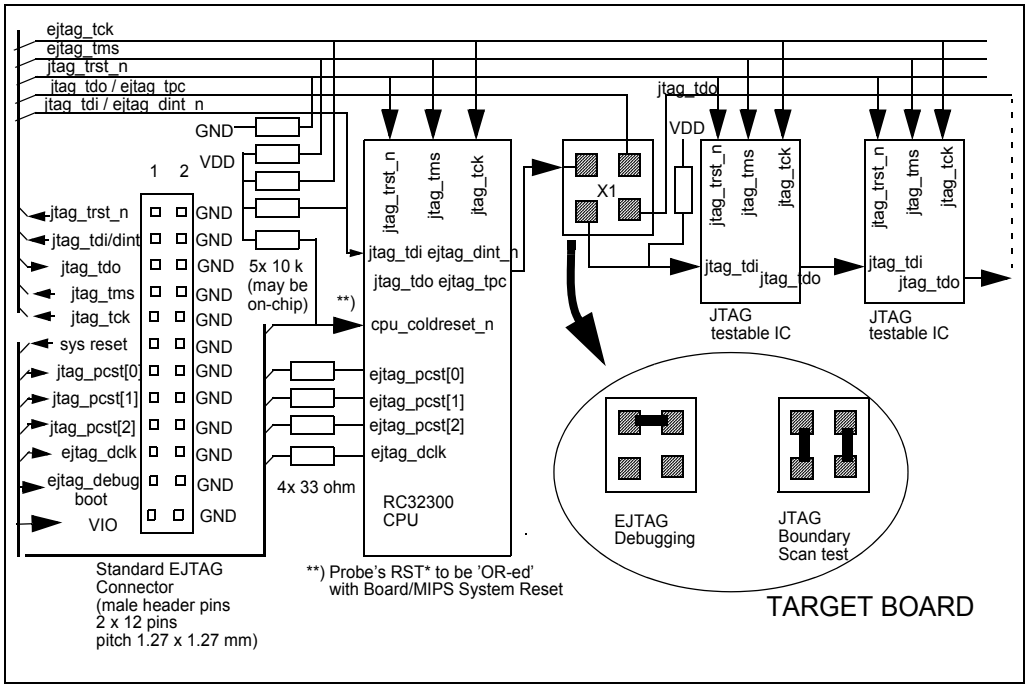


Figure 21.24 Application Diagram of Target Board and EJTAG Connection

Notes

Jumper block X1 on the Target Board provides the connection of the processor's jtag_tdo/ejtag_tpc signal to the EJTAG connector (during EJTAG debugging) or to the other boundary scan testable IC's on the board (during boundary scan test). This separates the (high speed) ejtag_tpc information from the other IC's during EJTAG emulation/debugging; after emulation/debugging is finished, the jumpers can be set such that the processor is part of the boundary scan test chain: jtag_tdo/ejtag_tpc outputs its serial data to the next following jtag_tdi input and the jtag_tdo of the last IC in the chain gets connected to the EJTAG connector. A JTAG/Boundary Scan tester can then be hooked to this connector (Pins 1-10 is sufficient in this case).

Since the EJTAG trace pins (ejtag_tpc, ejtag_pcst[2:0], ejtag_dclk) contain high speed data, the user shall take special care in the PCB layout of these signals. The EJTAG connector has to be placed close to the EJTAG pins of the processor chip; the PC Trace PCB tracks between connector and chip shall be short and preferably be of equal length.

The EJTAG Probe shall have a female connector that is plugged into this Target Connector. The EJTAG Probe Connector PCB may also contain a (fast) buffer for the high-speed trace signals; this external buffer shall be capable of driving the (short) flat cable to the EJTAG Probe box.

Hot Plug-In of the EJTAG Probe to Target System

In order to allow hot plug (connection while power on) the jtag_trst, jtag_tdi / ejtag_dint, jtag_tms and jtag_tck should be tri-state in the EJTAG Probe when the connection is made to target. In this way, the connection will not reset the target board by accident, and the input signals to the target could then be driven high to the right level when the Vdd is known.

Notes

RC32300 CPU Core Enhancements to MIPS II ISA

Notes

Introduction

The RC32300 execution unit implements the Enhanced MIPS-II ISA. A superset of MIPS II, these architectural enhancements include the addition of a MIPS-IV prefetch operation that incorporates various hint subfields, conditional move instructions that are MIPS-IV compatible, additional integer multiply unit instructions, and two new instructions designed to enhance the performance levels of certain DSP algorithms.

These features combine to make the CPU well suited to applications that require high bandwidth, rapid computation, and/or DSP capability. A discussion of each new integer unit feature implemented in the RC32334 follows. General instruction set information can be found in the *IDT MIPS Microprocessor Family Software Developer's Guide*.

Prefetch (PREF)

In general, PREF is an advisory instruction that may change the performance of the program but will not cause addressing related exceptions. If the PREF instruction raises an exception condition, the exception condition is ignored.

If an addressing-related exception condition is raised and ignored, no data will be prefetched. In such a case, if no data is prefetched, some action that is not architecturally-visible—such as writeback of a dirty cache line or invalidate a cache line (in the case of "ignorehit" hint)—might take place.

PREF will not generate a memory operation for a location with an uncached memory access type. As noted in Figure A.1, the hint field supplies information about the way the data is expected to be used. For data movement, the MIPS IV PREF instruction is implemented with multiple hints.

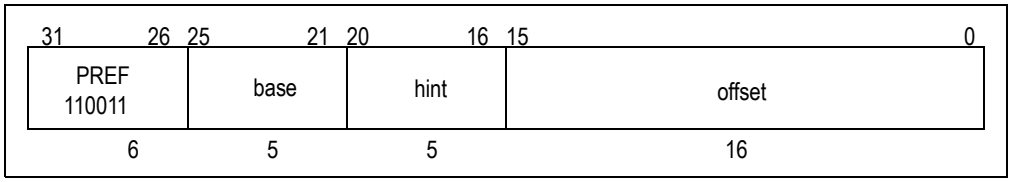


Figure A.1 Format of Prefetch Instruction

Format: PREF hint, offset(base)

Description: To form an effective byte address, PREF adds the 16-bit signed offset to the content of GPR base. It advises that data at the effective address may be used in the near future. The hint field supplies information about the way the data is expected to be used. The format of the Prefetch Instruction is shown in Figure A.1. Figure A.2 provides a diagram of the Prefetch operation flow.

Notes

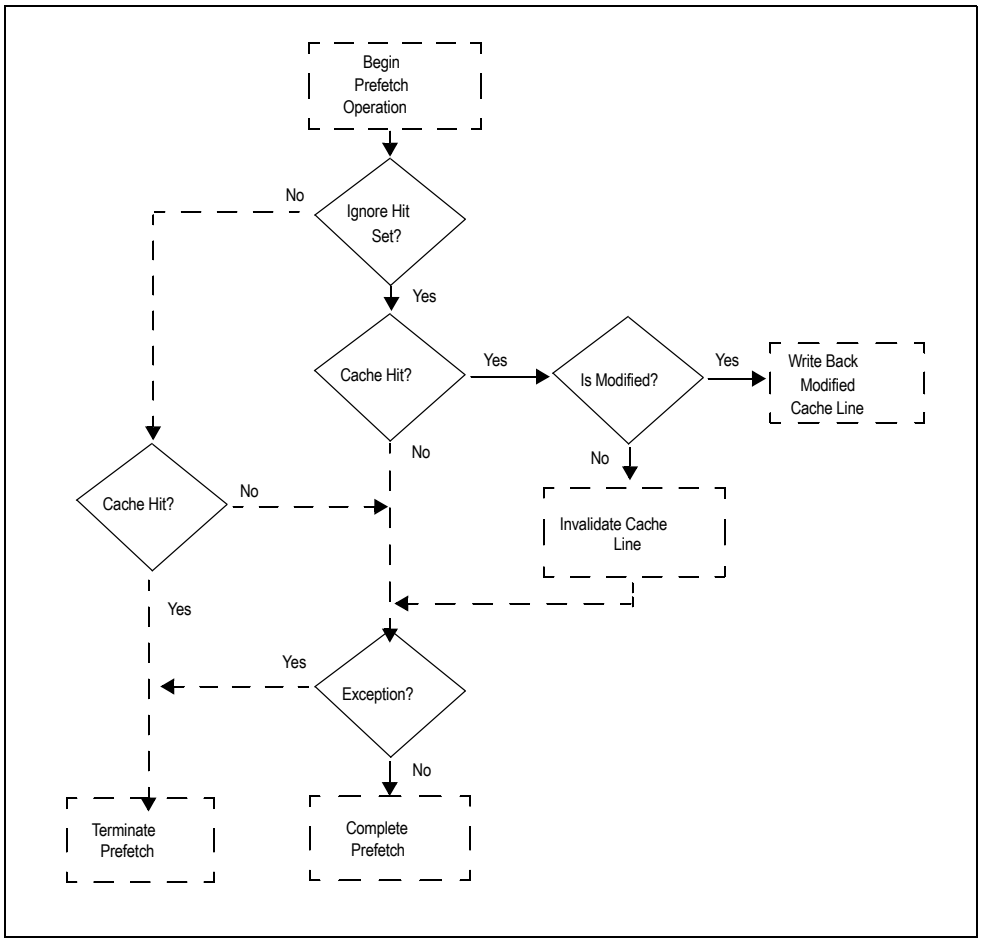


Figure A.2 Flowchart for Prefetch Operation

The defined hint values and prefetch actions are listed in Table A.1.

Value	Hint Field Name and Definition	Prefetch Action
0	Load Informs the CPU to process the PREF as if the cause were a cache miss on a load instruction. As such, the TLB coherency algorithm rules that apply to a load cache miss are applied. For example, if the TLB and chip were to support multi-processing, the resulting read could be marked as “coherent” or not, depending upon the translation.	Data is expected to be loaded (not modified). Fetch data as if for a load.
1	Store Informs the CPU to process the PREF as if the cause were a cache miss on a load instruction. As such, the rules with respect to coherency, write allocation, etc. may be applied to the resulting bus transaction.	Data is expected to be stored or modified. Fetch data as if for a store.
31	Ignore hit (Kernel Mode only) Causes the PREF to perform a cache refill, even if the target address currently hits in the cache. The MIPS-IV ISA allows PREF to revert to a NOP operation under exceptional conditions, etc., since the program will be semantically correct, although lower performance, if the cache miss processing occurs later. However, the “ignore hit” option carries an implicit invalidation of the current cache line. As such, even if the PREF/ignore-hit generates an exception, the cache line invalidate occurs when the PREF is encountered so that the program does run correctly later (that is, old cache contents are not used).	Invalidate the cache line and bring in the new data from memory regardless of the state of the valid bit.

Table A.1 Value of Hint Field for the Prefetch Instruction

Notes

Operation:

```
vAddr <-- GPR[base] + sign_extend(offset)
(pAddr, uncache) <-- Address Translation(vAddr, DATA, LOAD)
Prefetch(uncache, pAddr, vAddr, DATA, hint)
```

Exception: Reserved Instruction, if "Ignore hit" is used in User Mode.

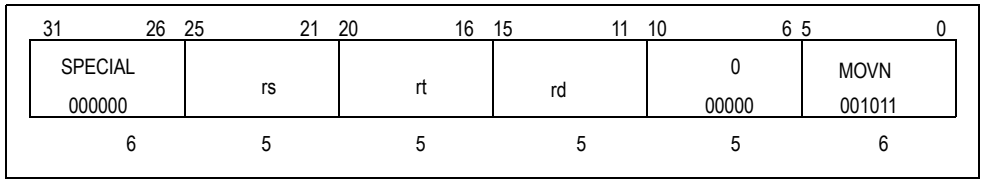
Elimination of 64-bit instructions

When an instruction requests 64-bit data operations, the RC32334 signals a trap. This includes both the MIPS-III 64-bit instructions and the MIPS-II 64-bit coprocessor operations. The trap signal occurs in both user and kernel modes.

Conditional Move Operations

In addition to the prefetch instruction, the RC32300 core implements the conditional move instructions found in the MIPS-IV architecture.

Move Conditional on Not Zero



Format: MOVN rd, rs, rt

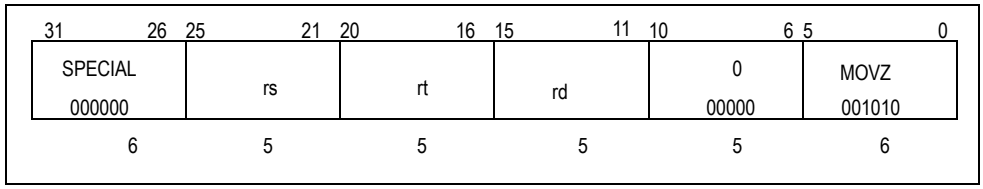
Description: If the value in rt is not equal to zero, then the content of rs is placed into rd.

Operation:

T: if GPR[rt] $\neq$ 0 then GPR[rd] <-- GPR[rs]

Exception: Reserved Instruction.

Move Conditional on Zero



Format: MOVZ rd, rs, rt

Description: If the value in rt is equal to zero, then the content of rs is placed into rd.

Operation:

T: if GPR[rt] = 0 then GPR[rd] <-- GPR[rs]

Exception: Reserved Instruction.

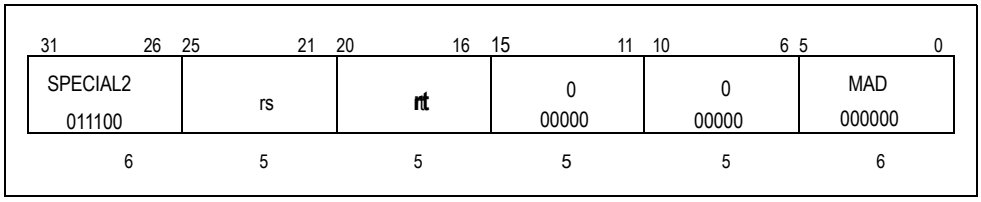
Instructions for DSP Support

The RC32300 CPU core adds new instructions to the MIPS II ISA, intended to enhance the performance of certain types of DSP algorithms. All of these extensions are supported in the Enhanced MIPS-II ISA.

Specifically, enhancements in the multiplier have been added to allow fast fused multiply-adds and multiply-subtracts. In addition, RC32300 CPU core adds the three operand multiply operations originally found in the 1st RC4650 and adds instructions to help normalize values (count-leading-1's or 0's).

Notes

Multiply Add



Format: MAD rs, rt

Description: The content of general registers rs and rt are multiplied— treating both operands as 32-bit two's complement values—and the result is added to HI/LO. Overflow exceptions do not occur under any circumstances.

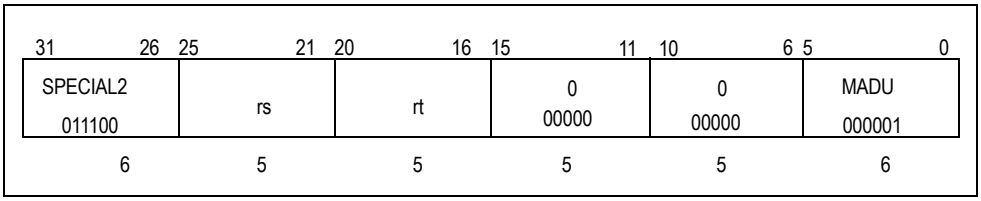
Once the operation is complete, the low-order word of the double result is loaded in LO, and the high-order word of the double result is loaded in HI.

Operation:

```
T:    temp <-- (HI || LO) + GPR[rs] * GPR[rt]
      LO <-- temp31..0
      HI <-- temp63..32
```

Exception: None

Multiply Add Unsigned



Format: MADU rs, rt

Description: The content of general registers rs and rt are multiplied, treating both operands as 32-bit unsigned values, and the result is added to HI/LO. No overflow exception occur under any circumstances.

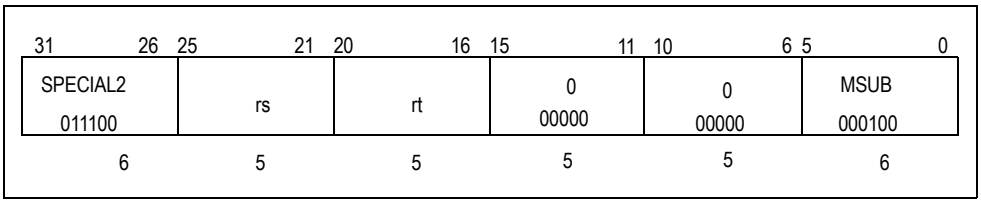
When the operation completes, the low-order word of the double result is loaded in LO, and the high-order word of the double result is loaded in HI. The instruction is not interlocked so any attempt to read HI/LO before the operation completes returns undefined value.

Operation:

```
T:    temp <-- (HI || LO) + (0||GPR[rs]) * (0||GPR[rt])
      LO <-- temp31..0
      HI <-- temp63..32
```

Exception: None

Multiply Subtract



Format: MSUB rs, rt

Notes

Description: The content of general registers rs and rt are multiplied, treating both operands as 32-bit two's complement values, and the result is subtracted from HI/LO. No overflow exception occur under any circumstances.

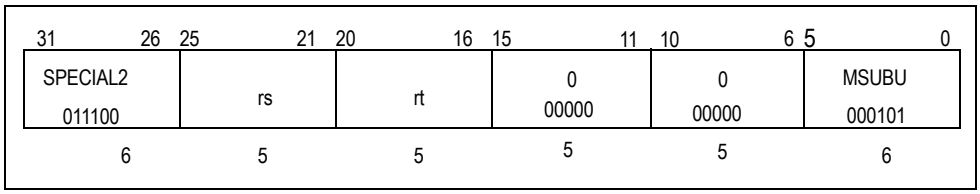
When the operation is complete, the low-order word of the double result is loaded in LO, and the high-order word of the double result is loaded into HI. The instruction is not interlocked so any attempt to read HI/LO before the operation completes returns undefined value.

Operation:

```
T:     temp <-- (HI || LO) - GPR[rs] * GPR[rt]
       LO <-- temp31..0
       HI <-- temp63..32
```

Exception: None

Multiply Subtract Unsigned



Format: MSUBU rs, rt

Description: The content of general registers rs and rt are multiplied, treating both operand as 32-bit unsigned values, and the result is subtracted from HI/LO. No overflow exception occur under any circumstances.

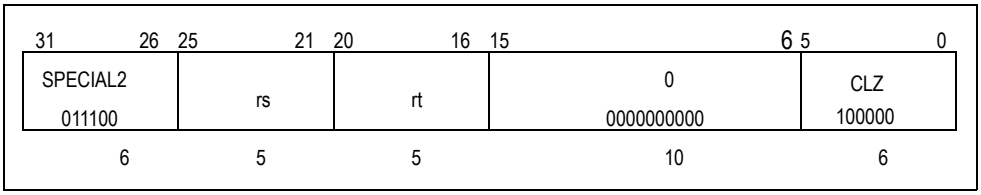
When the operation completes, the low-order word of the double result is loaded in LO, and the high-order word of the double result is loaded in HI. The instruction is not interlocked so any attempt to read HI/LO before the operation completes returns undefined value.

Operation:

```
T:     temp <-- (HI || LO) - (0||GPR[rs]) * (0||GPR[rt])
       LO <-- temp31..0
       HI <-- temp63..32
```

Exception: None

Count Leading Zeros



Format: CLZ rs, rt

Description: The content of general register rs is scanned from the most significant bit to the least significant bit, and the number of leading zeros is written into general register rt. If no bits were set in general register rs, i.e. rs=0, the content of general register rt is 32.

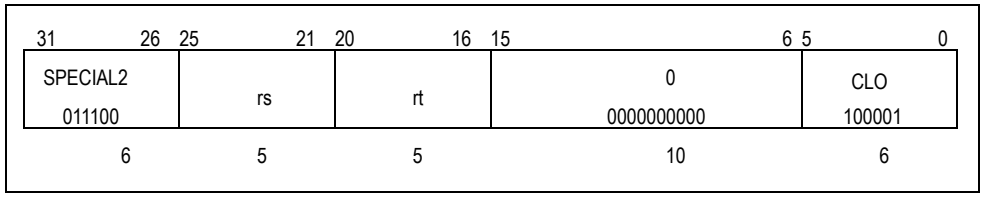
Operation:

```
T:     rt <-- Leading_zeros(rs)
```

Exception: None

Notes

Count Leading Ones



Format: CLO rs, rt

Description: The content of general register rs is scanned from most significant bit to least significant bit, the number of leading ones is written into general register rt. If no bits were cleared in general register rs, i.e. rs=0xffffffff, the content of general register rt is 32.

Operation:

```
T:      rt <-- Leading_ones(rs)
```

Exception: None

Opcode Map

Notes

Opcode

31..29 \ 28..26	0	1	2	3	4	5	6	7
0	SPECIAL	REGIMM	J	JAL	BEQ	BNE	BLEZ	BGTZ
1	ADDI	ADDIU	SLTI	SLTIU	ANDI	ORI	XORI	LUI
2	COP0	COP1	COP2	*	BEQL	BNEL	BLEZL	BGTZL
3	*	*	*	*	Special2	*	*	*
4	LB	LH	LWL	LW	LBU	LHU	LWR	*
5	SB	SH	SWL	SW	*	*	SWR	CACHE δ
6	LL	LWC1	LWC2	PREF	*	*	*	*
7	SC	SWC1	SWC2	*	*	*	*	*

SPECIAL function

5..3 \ 2..0	0	1	2	3	4	5	6	7
0	SLL	*	SRL	SRA	SLLV	*	SRLV	SRAV
1	JR	JALR	MOVZ	MOVN	SYSCALL	BREAK	SDBBP	SYNC
2	MFHI	MTHI	MFLO	MTLO	*	*	*	*
3	MULT	MULTU	DIV	DIVU	*	*	*	*
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	*	*	SLT	SLTU	*	*	*	*
6	TGE	TGEU	TLT	TLTU	TEQ	*	TNE	*
7	*	*	*	*	*	*	*	*

SPECIAL function2

5..3 \ 2..0	0	1	2	3	4	5	6	7
0	MAD	MADU	MUL	*	MSUB	MSUBU	*	*
1	*	*	*	*	*	*	*	*
2	*	*	*	*	*	*	*	*
3	*	*	*	*	*	*	*	*
4	CLZ	CLO	*	*	*	*	*	*
5	*	*	*	*	*	*	*	*
6	*	*	*	*	*	*	*	*
7	*	*	*	*	*	*	*	*

Notes

COPz rs

25,24 \ 23..21	0	1	2	3	4	5	6	7
0	MF	DMF	CF	γ	MT	DMT	CT	γ
1	BC	γ	γ	γ	γ	γ	γ	γ
2	CO							
3								

COPz rt

20..19 \ 18..16	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	γ	γ	γ	γ
1	γ	γ	γ	γ	γ	γ	γ	γ
2	γ	γ	γ	γ	γ	γ	γ	γ
3	γ	γ	γ	γ	γ	γ	γ	γ

CP0 Function

5..3 \ 2..0	0	1	2	3	4	5	6	7
0	ϕ	TLBR	TLBWI	ϕ	ϕ	ϕ	TLBWR	ϕ
1	TLBP	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
2	f	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
3	ERET	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	DRET
4	WAIT	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
5	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
6	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
7	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ

REGIMM rt

20..19 \ 18..16	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*	*	*	*
1	TGEI	TGEIU	TLTI	TLTIU	TEQI	TNEI	*	*
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*	*	*	*
3	*	*	*	*	*	*	*	*

The Timing of Cache Operations

Notes

Introduction

Cache holds a copy of recently read or written to memory data so that it can be quickly returned to the CPU. To double the effective cache-memory bandwidth, IDT CPUs implement separate on-chip instruction (I-cache) and data (D-cache) caches. Within the RC32334, both an I-cache and D-cache access can occur simultaneously; cache accesses take one processor clock to complete.

Information specific to the RC32334's cache organization and operation is provided in Chapter 7 of this manual.

Caveats About Cache Operations

- *All cycle counts are in processor cycles.*
- *All cache operations have a lower priority than cache misses, write backs and external requests. If the write back buffer contains unwritten data when a cache op is executed, the write back buffer will be retired before the cache op is started.*

If an instruction cache miss occurs at the same time a cache op is executed, the instruction cache miss will be handled first. Cache operations are mutually exclusive with respect to data cache misses. Before beginning any cache operation, external requests will be completed first.

- *For all data cache ops the cache op state machine waits for the store buffer and response buffer to empty before beginning the cache op. This can add 3 cycles to any data cache op if there is data in the response buffer or store buffer. The response buffer contains data from the last data cache miss that has not yet been written to the data cache. The store buffer contains delayed store data waiting to be written to the data cache.*
- *Cache ops of the form xxxx_Writeback_xxxx may perform a write back which will fill the write back buffer. Write backs can affect subsequent cache ops, since they will stall until the write back buffer is written back to memory. Cache ops which fill the write back buffer are noted as (writeback) in the following tables.*
- *All cycle counts are best case assuming no interference from the mechanisms described above.*

Cache Operations Tables

Table C.1 and Table C.2 show data cache and instruction cache operation's information. A detailed explanation of the Fill_I equation follows Table C.2.

Name	Operation	Number of Cycles
Index_Writeback_Invalidate_D	Examine the cache state and W bit of the primary data cache block at the index specified by the virtual address. If the state is not Invalid and the W bit is set, then write back the block to memory. The address to write is taken from the primary cache tag. Set cache state of primary cache block to Invalid.	10 cycles, if the cache line is clean. 12 cycles, if the cache line is dirty (Writeback).
Index_Load_Tag_D	Read the tag for the cache block at the specified index and place it into the TagLo CPO register, ignoring parity errors. Also load the data parity bits into the ECC register.	7 cycles.

Table C.1 Primary Data Cache Operations (Part 1 of 2)

Notes

Name	Operation	Number of Cycles
Index_Store_Tag_D	Write the tag for the cache block at the specified index from the TagLo and TagHi CPO registers	8 cycles.
Create_Dirty_Exclusive_D	This operation is used to avoid loading data needlessly from memory when writing new contents into an entire cache block. If the cache block does not contain the specified address, and the block is dirty, write it back to the memory. In all cases, set the cache block tag to the specified physical address and set the cache state to Dirty Exclusive.	10 cycles, for a cache hit. 13 cycles, for a cache miss if the cache line is clean. 15 cycles, for a cache miss if the cache line is dirty (Writeback).
Hit_Invalidate_D	If the cache block contains the specified address, mark the cache block invalid.	7 cycles, for a cache miss. 9 cycles, for a cache hit.
Hit_Writeback_Invalidate_D	If the cache block contains the specified address, write back the data if it is dirty and mark the cache block invalid.	7 cycles, for a cache miss. 12 cycles, for a cache hit if the cache line is clean. 14 cycles, for a cache hit if the cache line is dirty (Writeback).
Hit_Writeback_D	If the cache block contains the specified address, and the W bit is set, write back the data to memory and clear the W bit.	7 cycles, for a cache miss. 10 cycles, for a cache hit if the cache line is clean. 14 cycles, for a cache hit if the cache line is dirty (Writeback).

Table C.1 Primary Data Cache Operations (Part 2 of 2)

Name	Operation	Number of Cycles
Index_Invalidate_I	Set the cache state of the cache block to Invalid. Index_Invalidate_I writes the physical address of the cache operation into the tag when it clears the valid bit, which is different from the RC4000 family.	7 cycles.
Index_Load_Tag_I	Read the tag for the cache block at the specified index and place it into the TagLo CPO register, ignoring parity errors. Also load the data parity bits into the ECC register.	7 cycles.
Index_Store_Tag_I	Write the tag for the cache block at the specified index from the TagLo and TagHi CPO registers.	8 cycles.
Hit_Invalidate_I	If the cache block contains the specified address, mark the cache block invalid.	7 cycles for a cache miss. 9 cycles for a cache hit.
Fill_I	Fill the primary instruction cache block from memory. If the CE bit of the Status register is set, the contents of the ECC register are used instead of the computed parity bits for an addressed doubleword, when written to the instruction cache.	Cycle number must be calculated based on the system response to a memory access, because Fill_I causes an instruction cache refill from memory. The number of processor cycles for a Fill_I cache op is calculated as follows: Number_of_cycles_for_a_Fill_I_CacheOp = 10 + {0 - (SYSDIV - 1)} + (2 x SYSDIV) + (ML x SYSDIV) + (D x SYSDIV)
Hit_Writeback_I	If the cache block contains the specified address, write back the data unconditionally.	7 cycles, for a cache miss. 20 cycles, for a cache hit (Writeback).

Table C.2 Primary Instruction Cache Operations

Notes

Fill_I Equation Definitions

The following definitions apply to the Fill_I equation listed in Table C.2:

SYSDIV: Number of processor cycles per system cycle: range is between 2 and 8.

ML: Number of system cycles of memory latency, defined as the number of cycles the internal IP bus is driven by the external agent before the first word of data appears.

D: Number of system cycles required to return the block of data, defined as the number of cycles beginning when the first word of data appears on the internal IP bus and ending when the last word of data appears on the internal IP bus, inclusive.

Notes

RC32334/RC32332 Standby Mode Operation

Notes

Introduction

The Standby Mode operation is a means of reducing the internal core's power consumption when the CPU is in a "standby" state. In this section, the Standby Mode operation is explained.

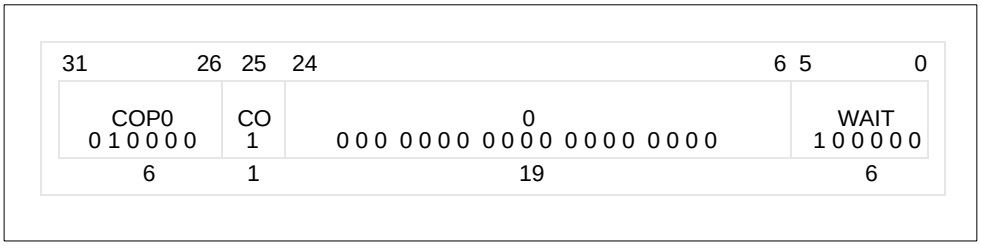
Power Management

The RC32334/RC32332 offers a number of features relevant to low-power systems, including low-power design, active power management, and a power-down operating mode.

Power Reduction Modes

The RISCore 32300 core is a static design, and products based on this core, such as the RC32334/RC32332, offer various power reduction modes. In addition, the RISCore 32300 supports a "Wait" instruction that is designed to signal the chip's other resources that execution and clocking should be halted.

The "Wait" instruction (illustrated and defined below) is used to halt the internal pipeline thus dramatically reducing the power consumption of the CPU.



Format: WAIT

Description: Used to halt the internal pipeline and reduce the power consumption of the CPU.

Operation:

```
T:      if AD bus is idle then
                StopPipeline
        endif
```

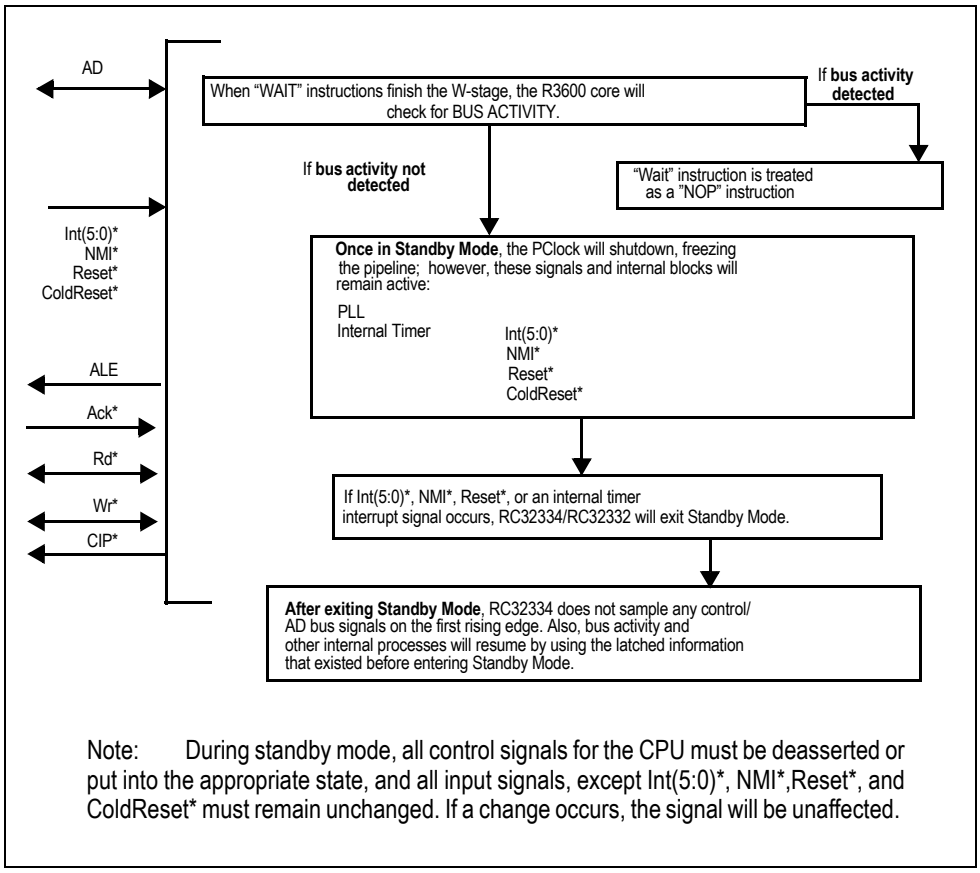
Exceptions: Coprocessor unusable exception.

Entering Standby Mode

To enter standby mode, first execute the WAIT instruction. When the WAIT instruction finishes the W pipe-stage, if the internal IP bus is currently idle, the internal clocks will shut down, thus freezing the pipeline. The PLL, internal timer, some of the input pin clocks (cpu_int_n[5:4,2:0], cpu_nmi_n, cpu_coldreset_n, internal cpu_int_n[3]) will continue to run. In the RC32334/RC32332, the system controller peripherals will continue to run. However, no DMA operations can occur while the CPU core is in standby mode.

If the conditions are not correct when the WAIT instruction finishes the W pipe-stage (such as the internal IP bus is not idle), the WAIT is treated as a NOP. Once the CPU is in standby mode, any interrupt—including the internally generated timer interrupt or the internal cpu_int_n[3]—will cause the CPU to exit standby mode. Figure D.1 illustrates the flow of the Standby Mode Operation.

Notes



Note: During standby mode, all control signals for the CPU must be deasserted or put into the appropriate state, and all input signals, except Int(5:0)*, NMI*,Reset*, and ColdReset* must remain unchanged. If a change occurs, the signal will be unaffected.

Figure D.1 Flowchart for Standby Mode Operation

Coprocessor 0 Hazards

Notes

Introduction

This appendix identifies the RC32300 CPU core specific coprocessor 0 hazards. Certain instruction combinations are not permitted because the results are unpredictable when combined with events such as pipeline delays, cache misses, interrupts and exceptions.

Most hazards result from instructions modifying and reading state in different pipeline stages. Such hazards are defined between pairs of instructions, not on any single instruction. Other hazards are associated with the restartability of instructions in the presence of exceptions.

Refer to the *IDT MIPS Microprocessor Family Software Developer's Guide* for information about MIPS ISA hazards.

List of Hazards

RC32334/RC32332 CP0 hazards are as follows:

- A mtc0 followed by a mfc0 is undefined. A one instruction delay between mtc0 and mfc0 is needed for proper operation. This rule applies when the destination of the first instruction is the same as the source of the second instruction. See Example #1 below.
- When DWatch is enabled, the two instructions immediately following may not be checked for a match with the watch value.
- When IWatch is enabled, the five instructions that follow may not be checked for a match with the I match value.
- When bit 23 of the Status register is changed, refills to set A may not be disabled until five instructions later.
- When bit 24 of the Status register is changed, refills to set A may not be disabled until three instructions later.
- Cannot clear UM, ERL, and EXL simultaneously. Must clear UM first, then ERL and EXL can be cleared simultaneously.
- A minimum of two NOP instructions should be inserted between the ERET instruction and the MTC0 instruction to ensure the EXL and ERL bits are changed correctly. See Example #2 below.

Example #1:

This instruction sequence will lead to an undefined result:

```
mtc0    r1, C0_SR
mfc0    k0, C0_SR
```

This instruction sequence will lead to the intended result:

```
mtc0    r1, C0_SR
mfc0    k0, C0_EPC
```

Example #2:

```
MTC0 CO_STATUS, R5
NOP
NOP
ERET
```

Notes

Integer Multiply Scheduling

Notes

Introduction

Integer multiply performance is substantially enhanced in the RC32334. The RC32300 CPU core adds a **MAD** instruction (multiply-accumulate, with **HI** and **LO** as the accumulator). Multiply performance is 2 cycles repeat, 3 cycles of latency for 16-bit operands (-2^{16} to $2^{16}-1$).

The **MAD** (multiply/add), **MADU** (multiply/add unsigned) MSUB (multiply/subtract) and MSUBU (multiply/subtract unsigned) are defined as follows, where **HI** and **LO** act as a 64-bit accumulator. These instructions do not trap on addition overflow.

MAD rs, rt	temp ← (HI|| LO) + rs * rt HI ← temp $_{63\,..\,32}$ LO ← temp $_{31\,..\,0}$

MADU rs, rt	temp ← (HI|| LO) $_{31\,..0}$) + (0|| rs) * (0|| rt) HI ← temp $_{63\,..\,32}$ LO ← temp $_{31\,..\,0}$

MSUB rs, rt	temp ← (HI || LO) - rs * rt HI ← temp $_{63\,..\,32}$ LO ← temp $_{31\,..\,0}$

MSUBU rs, rt	temp ← (HI || LO) - (0|| rs) * (0|| rt) HI ← temp $_{63\,..\,32}$ LO ← temp $_{31\,..\,0}$

In addition, the RC32300 CPU core implements another new multiply opcode that allows the multiply result to be returned directly to the primary register file:

MUL rd, rs, rt	temp ← rs $_{31\,...\,0}$ * rt $_{31\,..\,0}$ rd ← temp $_{31...\,0}$ HI ← undefined LO ← undefined

After executing this instruction, the **HI** and **LO** registers are undefined. For 16-bit operands, the latency of **MUL** is 3 cycles, with a repeat rate of 2 cycles. The **MUL** instruction will also unconditionally slip or stall for all but 2 cycles of its latency.

The performance of integer multiply and divide is summarized in Table F.1.

Notes

Opcodes	Condition	Latency	Repeat	Stall
MULT, MAD	$-2^{15} \leq rt \leq 2^{15}-1$	3	2	0
MULT, MAD	$rt < -2^{15}$ or $rt > 2^{15}-1$	4	3	0
MULTU, MADU	$0 \leq rt \leq 2^{16}-1$	3	2	0
MULTU, MADU	$rt > 2^{16}-1$	4	3	0
MUL	$-2^{15} \leq rt \leq 2^{15}-1$	3	2	1
	$rt < -2^{15}$ or $rt > 2^{15}-1$	4	3	2
DIV, DIVU	any	36	36	0

Table F.1 Integer Multiply and Divide Performance

As a special case, a MAD or MADU that is followed by a MUL instruction has one additional cycle of repeat above the value specified in the table.

In the RC4700, the MFLO and MFHI instructions do not make their results available immediately. If the RC4700 instruction references the MFLO/MFHI destination, then a 1-cycle slip will occur; however, on the RC32300 CPU core, the result is available immediately and there is no slip.

RC32332 Differences

Notes

Introduction

Generally, the information contained in this manual applies equally to both the RC32334 and the RC32332. Differences between the two devices are noted in this appendix and in occasional notes and footnotes throughout the manual.

The RC32332 is based on the same die as the RC32334, except that the RC32332 is housed in a 208 quad flat pack (QFP) package instead of the 256 ball grid array package used by the RC32334.

The QFP package option enables IDT to provide the solution at a lower cost point than the RC32334, but the reduced number of package connections means that certain features originally included in the RC32334 are reduced or removed from the RC32332.

Differences in Features

Table G.1 lists the differences in features between the RC32332 and RC32334.

	RC32332	RC32334
SDRAM bus interface	66 MHz	75 MHz
Memory address lines	23, 8MB maximum per CS	26, 64MB maximum per CS
PCI maximum frequency	50 MHz	66 MHz
On-chip PCI arbiter	2 slot	3 slot
DMA Controller	Flow control for Channel 0	Flow control for Channel 1 and 0
PIO Controller	8 PIO signals	16 PIO signals
TIMER Controller	—	1 external tc_n/gate_n signal
CPU interrupts	2 external	4 external
Number of UART channels	1	2
UART Controller	—	Modem control signals for Channel 0
Packaging	208 QFP	256 plastic BGA

Table G.1 Feature Set Comparison Between RC32332 and RC32334

Memory Controller

The RC32332 maps out fewer memory address lines—mem_addr[22:2] instead of mem_addr[25:2]. Therefore, with the RC32332, the maximum external memory size that can be supported for each individual chip select is 8MB.

PCI Controller On-chip Arbiter

The on-chip bus arbiter in the RC32332 supports two external bus masters: pci_req_n[0] and pci_req_n[2]. References in this manual to pci_req_n[1] do not apply to the RC32332.

PCI Controller Device ID

On the RC32332, it is recommended that the PCI Device ID be written as 0205h, either through the configuration register interface, or, if in the PCI Boot Mode, through the PCI Boot EEPROM. Using the recommended value will distinguish the controller from the RC32334. The default for the RC32332 is 204h, the same value as for the RC32334. For more details, see “Device ID Register” on page 12-26.

Notes

DMA Controller Flow Control

On the RC32332, there is one flow control signal, dma_ready_n[0] for DMA Channel 0. On the RC32334, there are two flow control signals, dma_ready_n[1:0] for DMA Channels 1 & 0. For more details, see Chapter 13.

PIO Controller Signals

The following 8 PIO signals are not available on the RC32332: uart_rts_n[0], uart_cts_n[0], uart_dsr_n[0], uart_dtr_n[0], uart_rx[1], uart_tx[1], timer_tc_n[0], and dma_ready_n[1]. For more details, see Chapter 15.

The following two tables summarize the differences between PIO pin names in the RC32334 and RC32332.

Register Bit	Main Function	Alternate Function RC32334	Alternate Function RC32332
31-12	Reserved	Reserved	Reserved
11	spi_mosi	PIO[10]	PIO[6]
10	spi_sck	PIO[9]	PIO[5]
9	spi_ss_n	PIO[8]	PIO[4]
8	spi_miso	PIO[7]	PIO[3]
7	uart_rx[0]	PIO[6]	PIO[2]
6	uart_tx[0]	PIO[5]	PIO[1]
5	uart_rx[1]	PIO[4]	Reserved
4	uart_tx[1]	PIO[3]	Reserved
3	timer_tc_n[0]	PIO[2]	Reserved
2	Reserved	Reserved	Reserved
1	dma_ready_n[0]	PIO[1]	PIO[0]
0	dma_ready_n[1]	PIO[0]	Reserved

Table G.2 PIO [Data/Direction/Function Select] Register 0 Comparison

Register Bit	Main Function	Alternate Function RC32334	Alternate Function RC32332
31-5	Reserved	Reserved	Reserved
4	uart_cts_n[0]	PIO[15]	Reserved
3	uart_dsr_n[0]	PIO[14]	Reserved
2	uart_dtr_n[0]	PIO[13]	Reserved
1	uart_rts_n[0]	PIO[12]	Reserved
0	pci_eeprom_cs	PIO[11]	PIO[7]

Table G.3 PIO [Data/Direction/Function Select] Register 1 Comparison

Notes

TIMER Controller Signal

On the RC32332, the timer overflow/gate signal, timer_tc_n[0] is not present. For more details, see Chapter 16.

Interrupt Lines

The RC32332 maps out one less interrupt line to the external pads. The RC32332 has two external interrupt lines available (cpu_int_n[1:0]) in addition to the NMI line. All references in this manual to the additional interrupt lines in the RC32334 (cpu_int_n[5:4], cpu_int_n[2]) do not apply to the RC32332.

UART Interface

The RC32332 has only one serial port (UART0). All features in this user manual referencing UART1 should be ignored if the designer is planning to use the RC32332. Additionally, for UART0, all of the modem signals that were bonded out to the external pads in the RC32334—Request to Send (RTS), Clear to Send (CTS), Data Terminal Ready (DTR), and Data Set Ready (DSR)—are not accessible on the RC32332 pins. Therefore, the programming of these bits in the modem control registers does not perform any usable function in the RC32332.

Internal Bus Interface SysID Register

The value for the RC32332 that is programmed in bits 19:8 is 004h. For more details, refer to “SysID Register” on page 8-14.

JTAG DEVICE_ID Register

The value for the RC32332 that is programmed in the Part Number field, bits 27:12, is 001Ah. For more details, see section DEVICEID in Chapter 20.

JTAG Boundary Scan Cells

The RC32332 has 303 boundary scan cells as described in its BSDL file. The RC32334 has 330 boundary scan cells, as described in its BSDL file.

Electrical / Pinout

See the RC32332 data sheet for information on the AC, DC, and thermal characteristics, and device pinout.

Pin Description Table

The following table lists the pins provided on the RC32332. Note that those pin names followed by "_n" are active-low signals. All external pull-ups and pull-downs require 10 kΩ resistor.

<table>
<tr><th>Name</th><th>Type</th><th>Reset State Status</th><th>Drive Strength Capability</th><th>Description</th></tr>
<tr><td colspan="5">Local System Interface</td></tr>
<tr><td>mem_data[31:0]</td><td>I/O</td><td>Z</td><td>High</td><td>Local system data bus
Primary data bus for memory. I/O and SDRAM.</td></tr>
<tr><td>mem_addr[22:2]</td><td>I/O</td><td>[25:10] Z
[9:2] L</td><td>[22:16] Low
[15:2] High</td><td>Memory Address Bus
These signals provide the Memory or DRAM address, during a Memory or DRAM bus transaction. During each word data, the address increments either in linear or sub-block ordering, depending on the transaction type. The table below indicates how the memory write enable signals are used to address discrete memory port width types.
<table>
<tr><th>Port Width</th><th>Pin Signals
mem_we_n[3]</th><th>mem_we_n[2]</th><th>mem_we_n[1]</th><th>mem_we_n[0]</th></tr>
<tr><td>DMA (32-bit)</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr>
<tr><td>32-bit</td><td>mem_we_n[3]</td><td>mem_we_n[2]</td><td>mem_we_n[1]</td><td>mem_we_n[0]</td></tr>
<tr><td>16-bit</td><td>Byte High Write Enable</td><td>mem_addr[1]</td><td>Not Used
(Driven Low)</td><td>Byte Low Write Enable</td></tr>
<tr><td>8-bit</td><td>Not Used
(Driven High)</td><td>mem_addr[1]</td><td>mem_addr[0]</td><td>Byte Write Enable</td></tr>
</table>
mem_addr[22] Alternate function: reset_boot_mode[1].
mem_addr[21] Alternate function: reset_boot_mode[0].
mem_addr[20] Alternate function: reset_pci_host_mode.
mem_addr[19] Alternate function: modebit [9].
mem_addr[18] Alternate function: modebit [8].
mem_addr[17] Alternate function: modebit [7].
mem_addr[15] Alternate function: sdram_addr[15].
mem_addr[14] Alternate function: sdram_addr[14].
mem_addr[13] Alternate function: sdram_addr[13].
mem_addr[11] Alternate function: sdram_addr[11].
mem_addr[10] Alternate function: sdram_addr[10].
mem_addr[9] Alternate function: sdram_addr[9].
mem_addr[8] Alternate function: sdram_addr[8].
mem_addr[7] Alternate function: sdram_addr[7].
mem_addr[6] Alternate function: sdram_addr[6].
mem_addr[5] Alternate function: sdram_addr[5].
mem_addr[4] Alternate function: sdram_addr[4].
mem_addr[3] Alternate function: sdram_addr[3].
mem_addr[2] Alternate function: sdram_addr[2]</td></tr>
<tr><td>mem_cs_n[5:0]</td><td>Output</td><td>H</td><td>Low</td><td>Memory Chip Select Negated Recommend external pull-up.
Signals that a Memory Bank is actively selected.</td></tr>
<tr><td>mem_oe_n</td><td>Output</td><td>H</td><td>High</td><td>Memory Output Enable Negated Recommend external pull-up.
Signals that a Memory Bank can output its data lines onto the cpu_ad bus.</td></tr>
<tr><td>mem_we_n[3:0]</td><td>Output</td><td>H</td><td>High</td><td>Memory Write Enable Negated Bus
Signals which bytes are to be written during a memory transaction. Bits act as Byte Enable and mem_addr[1:0] signals for 8-bit or 16-bit wide addressing.</td></tr>
<tr><td>mem_wait_n</td><td>Input</td><td></td><td>—</td><td>Memory Wait Negated Requires external pull-up.
SRAM/IOI/IOM modes: Allows external wait-states to be injected during the last cycle before data is sampled.
DPM (dual-port) mode: Allows dual-port busy signal to restart memory transaction.
Alternate function: sdram_wait_n.</td></tr>
</table>

Table 21.26 Pin Description for RC32332 (Part 1 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
mem_245_oe_n	Output	H	Low	**Memory FCT245 Output Enable Negated** Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to a memory or I/O bank.
mem_245_dt_r_n	Output	Z	High	**Memory FCT245 Direction Xmit/Rcv Negated** Recommend external pull-up. Alternate function: cpu_dt_r_n. See CPU Core Specific Signals below.
output_clk	Output	cpu-masterclk	High	**Output Clock** Optional clock output.
PCI Interface				
pci_ad[31:0]	I/O	Z	PCI	**PCI Multiplexed Address/Data Bus** Address driven by Bus Master during initial frame_n assertion, and then the Data is driven by the Bus Master during writes; or the Data is driven by the Bus Slave during reads.
pci_cbe_n[3:0]	I/O	Z	PCI	**PCI Multiplexed Command/Byte Enable Bus** Command (not negated) Bus driven by the Bus Master during the initial frame_n assertion. Byte Enable Negated Bus driven by the Bus Master during the data phase(s).
pci_par	I/O	Z	PCI	**PCI Parity** Even parity of the pci_ad[31:0] bus. Driven by Bus Master during Address and Write Data phases. Driven by the Bus Slave during the Read Data phase.
pci_frame_n	I/O	Z	PCI	**PCI Frame Negated** Driven by the Bus Master. Assertion indicates the beginning of a bus transaction. De-assertion indicates the last datum.
pci_trdy_n	I/O	Z	PCI	**PCI Target Ready Negated** Driven by the Bus Slave to indicate the current datum can complete.
pci_irdy_n	I/O	Z	PCI	**PCI Initiator Ready Negated** Driven by the Bus Master to indicate that the current datum can complete.
pci_stop_n	I/O	Z	PCI	**PCI Stop Negated** Driven by the Bus Slave to terminate the current bus transaction.
pci_idsel_n	Input		—	**PCI Initialization Device Select** Uses pci_req_n[2] pin. See the PCI subsection.
pci_perr_n	I/O	Z	PCI	**PCI Parity Error Negated** Driven by the receiving Bus Agent 2 clocks after the data is received, if a parity error occurs.
pci_serr_n	I/O Open-collector	Z	PCI	**System Error** External pull-up resistor is required. Driven by any agent to indicate an address parity error, data parity during a Special Cycle command, or any other system error.
pci_clk	Input		—	**PCI Clock** Clock for PCI Bus transactions. Uses the rising edge for all timing references.
pci_rst_n	Input	L	—	**PCI Reset Negated** Host mode: Resets all PCI related logic. Satellite mode: with boot from PCI mode: Resets all PCI related logic and also warm resets the 32332.
pci_devsel_n	I/O	Z	PCI	**PCI Device Select Negated** Driven by the target to indicate that the target has decoded the present address as a target address.
pci_req_n[2]	Input	Z	—	**PCI Bus Request #2 Negated** Requires external pull-up. Host mode: pci_req_n[2] is an input indicating a request from an external device. Satellite mode: used as pci_idsel pin which selects this device during a configuration read or write. Alternate function: pci_idsel (satellite).
pci_req_n[0]	I/O	Z	High	**PCI Bus Request #0 Negated** Requires external pull-up for burst mode. Host mode: pci_req_n[0] is an input indicating a request from an external device. satellite mode: pci_req_n[0] is an output indicating a request from this device.

Table 21.26 Pin Description for RC32332 (Part 2 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
pci_gnt_n[2]	Output	Z[1]	High	**PCI Bus Grant #2 Negated** Recommend external pull-up. Host mode: pci_gnt_n[2] is an output indicating a grant to an external device. Satellite mode: pci_gnt_n[2] is used as the pci_inta_n output pin. External pull-up is required. Alternate function: pci_inta_n (satellite).
pci_gnt_n[1]	I/O	X for 1 pci clock then H[2]	High	**PCI Bus Grant #1 Negated** Recommend external pull-up. Host mode: not used. Satellite mode: Used as pci_eprom_cs output pin for Serial Chip Select for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: pci_eeprom_cs (satellite). 2nd Alternate function: PIO[7].
pci_gnt_n[0]	I/O	Z	High	**PCI Bus Grant #0 Negated** Host mode: pci_gnt_n[0] is an output indicating a grant to an external device. Recommend external pull-up. Satellite mode: pci_gnt_n[0] is an input indicating a grant to this device. Requires external pull-up.
pci_inta_n	Output Open-collector	Z	PCI	**PCI Interrupt #A Negated** Uses pci_gnt_n[2]. See the PCI subsection.
pci_lock_n	Input		—	**PCI Lock Negated** Driven by the Bus Master to indicate that an exclusive operation is occurring.

[1] Z in host mode; L in satellite non-boot mode; Z in satellite boot mode.
[2] H in host mode; L in satellite non-boot mode; L in satellite boot mode.

SDRAM Control Interface

sdram_addr_12	Output	L	High	**SDRAM Address Bit 12 and Precharge All** SDRAM mode: Provides SDRAM address bit 12 (10 on the SDRAM chip) during row address and "precharge all" signal during refresh, read and write command.
sdram_ras_n	Output	H	High	**SDRAM RAS Negated** SDRAM mode: Provides SDRAM RAS control signal to all SDRAM banks.
sdram_cas_n	Output	H	High	**SDRAM CAS Negated** SDRAM mode: Provides SDRAM CAS control signal to all SDRAM banks.
sdram_we_n	Output	H	High	**SDRAM WE Negated** SDRAM mode: Provides SDRAM WE control signal to all SDRAM banks.
sdram_cke	Output	H	High	**SDRAM Clock Enable** SDRAM mode: Provides clock enable to all SDRAM banks.
sdram_cs_n[3:0]	Output	H	High	**SDRAM Chip Select Negated Bus** Recommend external pull-up. SDRAM mode: Provides chip select to each SDRAM bank. SODIMM mode: Provides upper select byte enables [7:4].
sdram_s_n[1:0]	Output	H	High	**SDRAM SODIMM Select Negated Bus** SDRAM mode: Not used. SDRAM SODIMM mode: Upper and lower chip selects.
sdram_bemask_n [3:0]	Output	H	High	**SDRAM Byte Enable Mask Negated Bus (DQM)** SDRAM mode: Provides byte enables for each byte lane of all DRAM banks. SODIMM mode: Provides lower select byte enables [3:0].
sdram_245_oe_n	Output	H	Low	**SDRAM FCT245 Output Enable Negated** Recommend external pull-up. SDRAM mode: Controls output enable to optional FCT245 transceiver bank by asserting during both reads and writes to any DRAM bank.
sdram_245_dt_r_n	Output	Z	High	**SDRAM FCT245 Direction Transmit/Receive** Recommend external pull-up. Uses cpu_dt_r_n. See CPU Core Specific Signals below.

Table 21.26 Pin Description for RC32332 (Part 3 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
On-Chip Peripherals				
dma_ready_n[0]	I/O	Z	Low	**DMA Ready Negated Bus** Requires external pull-up. Ready mode: Input pin for general purpose DMA channel 0 that can initiate the next datum in the current DMA descriptor frame. Done mode: Input pin for general purpose DMA channel 0 that can terminate the current DMA descriptor frame. dma_ready_n[0] 1st Alternate function PIO[0]; 2nd Alternate function: dma_done_n[0].
pio[7:0]	I/O	See related pins	Low	**Programmable Input/Output** General purpose pins that can each can be configured as a general purpose input or general purpose output. These pins are multiplexed with other pin functions: pci_gnt_n[1], spi_mosi, spi_miso, spi_sck, spi_ss_n, uart_rx[0], uart_tx[0], dma_ready_n[0]. Note that pci_gnt_n[1], spi_mosi, spi_sck, and spi_ss_n default to outputs at reset time. The others default to inputs.
uart_rx[0]	I/O	Z	Low	**UART Receive Data Bus** UART mode: UART channel receives data. uart_rx[0] Alternate function: PIO[2].
uart_tx[0]	I/O	Z	Low	**UART Transmit Data** UART mode: UART channel send data. Note that this pin defaults to an input at reset time and must be programmed via the PIO interface before being used as a UART output. uart_tx[0] Alternate function: PIO[1].
spi_mosi	I/O	L	Low	**SPI Data Output** Serial mode: Output pin from RC32332 as an Input to a Serial Chip for the Serial data input stream. In PCI satellite mode, acts as an Output pin from RC32332 that connects as an Input to a Serial Chip for the Serial data input stream for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[6]. 2nd Alternate function: pci_eeprom_mdo.
spi_miso	I/O	Z	Low	**SPI Data Input** Serial mode: Input pin to RC32332 from the Output of a Serial Chip for the Serial data output stream. In PCI satellite mode, acts as an Input pin from RC32332 that connects as an output to a Serial Chip for the Serial data output stream for loading PCI Configuration Registers in the RC32332 Reset Initialization Vector PCI boot mode. 1st Alternate function: PIO[3]. 2nd Alternate function: pci_eeprom_mdi.
spi_sck	I/O	L	Low	**SPI Clock** Serial mode: Output pin for Serial Clock. In PCI satellite mode, acts as an Output pin for Serial Clock for loading PCI Configuration Registers in the RC323332 Reset Initialization Vector PCI boot mode. Defaults to the output direction at reset time. 1st Alternate function: PIO[5]. 2nd Alternate function: pci_eeprom_sk.
spi_ss_n	I/O	H	Low	**SPI Chip Select** Output pin selecting the serial protocol device as opposed to the PCI satellite mode EEPROM device. Alternate function: PIO[4]. Defaults to the output direction at reset time.
CPU Core Specific Signals				
cpu_nmi_n	Input		—	**CPU Non-Maskable Interrupt** Requires external pull-up. This interrupt input is active low to the CPU.
cpu_masterclk	Input		—	**CPU Master System Clock** Provides the basic system clock.
cpu_int_n[1:0]	Input		—	**CPU Interrupt** Requires external pull-up. These interrupt inputs are active low to the CPU.

Table 21.26 Pin Description for RC32332 (Part 4 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
cpu_coldreset_n	Input	L	—	**CPU Cold Reset** This active-low signal is asserted to the RC32332 after V_{CC} becomes valid on the initial power-up. The Reset initialization vectors for the RC32332 are latched by cold reset.
cpu_dt_r_n	Output	Z	—	**CPU Direction Transmit/Receive** This active-low signal controls the DT/R pin of an optional FCT245 transceiver bank. It is asserted during read operations. 1st Alternate function: mem_245_dt_r_n. 2nd Alternate function: sdram_245_dt_r_n.
JTAG Interface Signals				
jtag_tck	Input		—	**JTAG Test Clock** Requires external pull-down. An input test clock used to shift into or out of the Boundary-Scan register cells. jtag_tck is independent of the system and the processor clock with nominal 50% duty cycle.
jtag_tdi, ejtag_dint_n	Input		—	**JTAG Test Data In** Requires an external pull-up on the board. On the rising edge of jtag_tck, serial input data are shifted into either the Instruction or Data register, depending on the TAP controller state. During Real Mode, this input is used as an interrupt line to stop the debug unit from Real Time mode and return the debug unit back to Run Time Mode (standard JTAG). Requires an external pull-up on the board. This pin is also used as the ejtag_dint_n signal in the EJTAG mode.
jtag_tdo, ejtag_tpc	Output	Z	High	**JTAG Test Data Out** The jtag_tdo is serial data shifted out from instruction or data register on the falling edge of jtag_tck. When no data is shifted out, the jtag_tdo is tri-stated. During Real Time Mode, this signal provides a non-sequential program counter at the processor clock or at a division of processor clock. This pin is also used as the ejtag_tpc signal in the EJTAG mode.
jtag_tms	Input		—	**JTAG Test Mode Select** Requires external pull-up. The logic signal received at the jtag_tms input is decoded by the TAP controller to control test operation. jtag_tms is sampled on the rising edge of the jtag_tck.
jtag_trst_n	Input	L	—	**JTAG Test Reset** When neither JTAG nor EJTAG are being used, jtag_trst_n must be driven or pulled low, or the jtag_tms/ejtag_tms signals must be pulled up and jtag_clk actively clocked.
ejtag_dclk	Output	Z	—	**EJTAG Test Clock** Processor Clock. During Real Time Mode, this signal is used to capture address and data from the ejtag_tpc signal at the processor clock speed or any division of the internal pipeline.
ejtag_pcst[2:0]	I/O	Z	Low	**EJTAG PC Trace Status Information** 111 (STL) Pipe line Stall 110 (JMP) Branch/Jump forms with PC output 101 (BRT) Branch/Jump forms with no PC output 100 (EXP) Exception generated with an exception vector code output 011 (SEQ) Sequential performance 010 (TST) Trace is outputted at pipeline stall time 001 (TSQ) Trace trigger output at performance time 000 (DBM) Run Debug Mode Alternate function: modebit[2:0].
ejtag_debugboot	Input		—	**EJTAG DebugBoot** Requires an external pull-down. The ejtag_debugboot input is used during reset and forces the CPU core to take a debug exception at the end of the reset sequence instead of a reset exception. This enables the CPU to boot from the ICE probe without having the external memory working. This input signal is level sensitive and is not latched internally. This signal will also set the JtagBrk bit in the JTAG_Control_Register[12].

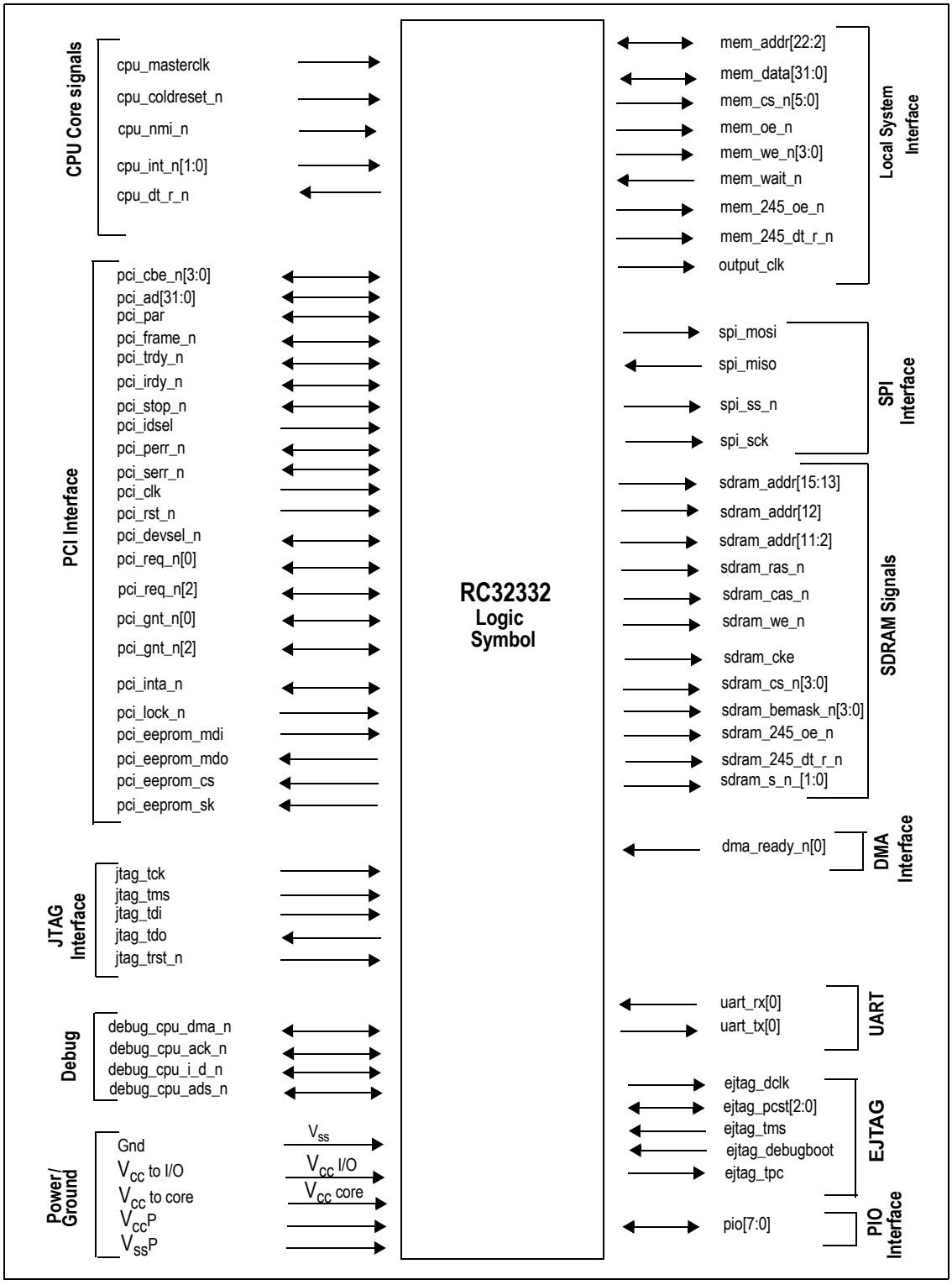
Table 21.26 Pin Description for RC32332 (Part 5 of 6)

Name	Type	Reset State Status	Drive Strength Capability	Description
ejtag_tms	Input		—	**EJTAG Test Mode Select** Requires an external pull-up. The ejtag_tms is sampled on the rising edge of jtag_tck.
Debug Signals				
debug_cpu_dma_n	I/O	Z	Low	**Debug CPU versus DMA Negated** Assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from the CPU. De-assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction was generated from DMA. Alternate function: modebit[6].
debug_cpu_ack_n	I/O	Z	Low	**Debug CPU Acknowledge Negated** Indicates either a data acknowledge to the CPU or DMA. Alternate function: modebit[4].
debug_cpu_ads_n	I/O	Z	Low	**Debug CPU Address/Data Strobe Negated** Assertion indicates that either a CPU or a DMA transaction is beginning and that the mem_data[31:4] bus has the current block address. Alternate function: modebit[5].
debug_cpu_i_d_n	I/O	Z	Low	**Debug CPU Instruction versus Data Negated** Assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU or DMA data transaction. De-assertion during debug_cpu_ads_n assertion or debug_cpu_ack_n assertion indicates transaction is a CPU instruction transaction. Alternate function: modebit[3].

Table 21.26 Pin Description for RC32332 (Part 6 of 6)

Logic Diagram

The logic diagram of the RC32332 differs from that of the RC32334.



Index

R

V

W