IDT79RC32364 RISController^{TM}
Advanced Architecture
32-bit Embedded Microprocessor,
User's Reference Manual

Version 1.1
April 1999

6024 Silver Creek Valley Road, San Jose, California 95138
Telephone: (800) 345-7015 • (408) 284-8200 • FAX: (408) 284-2775
Printed in U.S.A.

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.

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 RC32364, the first 32-bit low-cost, low-powered member of the Integrated Device Technology (IDT) RISController Series of Embedded Microprocessors. Operational overviews, functional descriptions and diagrams are provided in each chapter to assist system developers in obtaining optimum device performance.

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, “RC32364 Device Overview,” provides an introduction to the performance capabilities of the 32-bit RC32364 microprocessor. Included in this chapter are performance and device overviews, a list of features and a block diagram. Chapter 1 also includes an explanation on the Byte ordering conventions used for this device.

Chapter 2, “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 3, “CPU Pipeline Architecture,” discusses pipeline features as well as interlock and exception handling of the device’s RISCore32300.

Chapter 4, “Memory Management,” contains a discussion on the virtual-to-physical address translation technique, TLB management, and operation modes for the RC32364. Register formats and field description tables are also provided in this chapter.

Chapter 5, “CPU Exception Processing,” defines and describes the various exception types and handling processes for the RC32364. Also provided in this chapter are the CPO register formats, their field descriptions and general exception handling flowcharts.

Chapter 6, “Cache Organization, Operation and Coherency,” includes a general discussion on the operation of cache as well as the more specific cache attributes of the RC32364. Flowcharts and various diagrams are provided to clarify the concepts discussed in this chapter.

Chapter 7, “Processor Signal Descriptions,” contains easy reference tables on the various interface signals used by and in conjunction with the RC32364 processor. Functionally grouped, each table provides signal information such as name, definition, direction, and description.

Chapter 8, “Clocking, Reset and Initialization Interfaces,” discusses the basic system clocks used by the processor. A general overview on the Phase-locked loop operation and power management features are also introduced in this chapter.

Chapter 9, “Bus Interface Overview,” provides explanations on the interface control registers used by the RC32364. Tables containing field descriptions are included as well as timing diagrams for the various addressing, interfacing, and DMA operations.

Appendix A, “**RC32364 Enhancements to MIPS 32 ISA**,” describes the MIPS 32 enhancements implemented on the RC32364.

Appendix B, “RC32364 Opcode Map,” provides the opcode map for this device.

Appendix C, “The Timing of Cache Operations,” lists primary cache operation cycle times.

Notes

Appendix D, “RC32364 Standby Mode Operation,” provides a flowchart and explanation of the standby mode operation implemented in the RC32364 processor.

Appendix E, “Coprocessor 0 Hazards,” identifies the coprocessor 0 hazards for the RC32364.

Appendix F, “Integer Multiply Scheduling,” defines the multiplication performance enhancements of the RC32364.

Appendix G, “EJTAG (In-circuit Emulator) Interface,” describes the protocols and capabilities of the serial interface that enables access to internal CPU units. This allows the programmer visibility of register and cache status information and control over the CPU’s operation, to simplify the system debug process.

Revision History

May 1998: Initial publication.

April 2000: Second edition.

September 5, 2000: Revised Appendix G: EJTAG Interface.

Table of Contents

Notes

About This Manual

1 RC32364 Device Overview

2 CPU Instruction Set Overview

Notes

Notes

7 Processor Signal Descriptions

8 Clocking, Reset and Initialization Interfaces

9 Bus Interface Overview

Notes

Notes

Appendix F Integer Multiply Scheduling

Appendix G EJTAG (In-circuit Emulator) Interface

Notes

Notes

List of Tables

Notes

Notes

List of Figures

Notes

Notes

Notes

Notes

RC32364 Device Overview

Notes

Introduction

Targeted to a variety of software intensive embedded applications, the RC32364 is a member of the Integrated Device Technology, Inc. (IDT) RISController series of Embedded Microprocessors. The RC32364 continues IDT's tradition of high-performance through high-speed pipelines, high-bandwidth caches and bus interface, MIPS application specific architectural extensions, and careful attention to efficient control. IDT's proprietary RISCore32300 CPU core provides low-powered operations, while maintaining high performance.

The RC32364 supports a wide variety of embedded processor-based applications, such as internetworking equipment (low-end routers and switches), telecommunications equipment (cellular base stations) and consumer multimedia game systems.

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 1.1 and Figure 1.2.

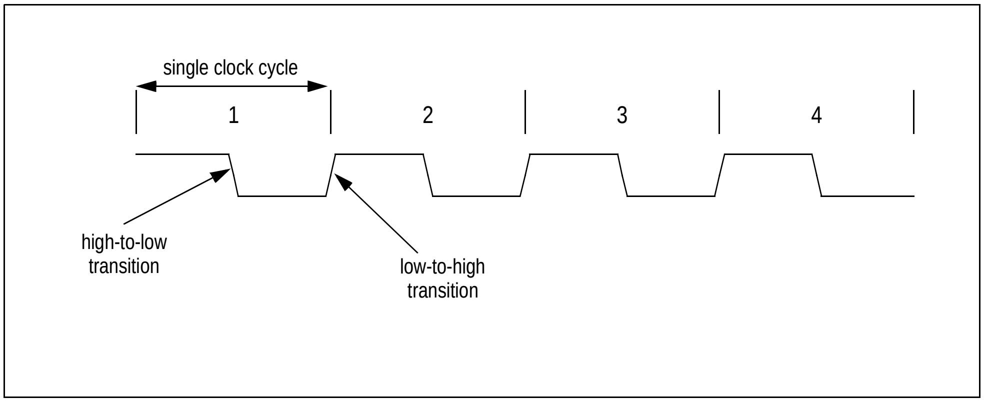


Figure 1.1 Signal Transitions

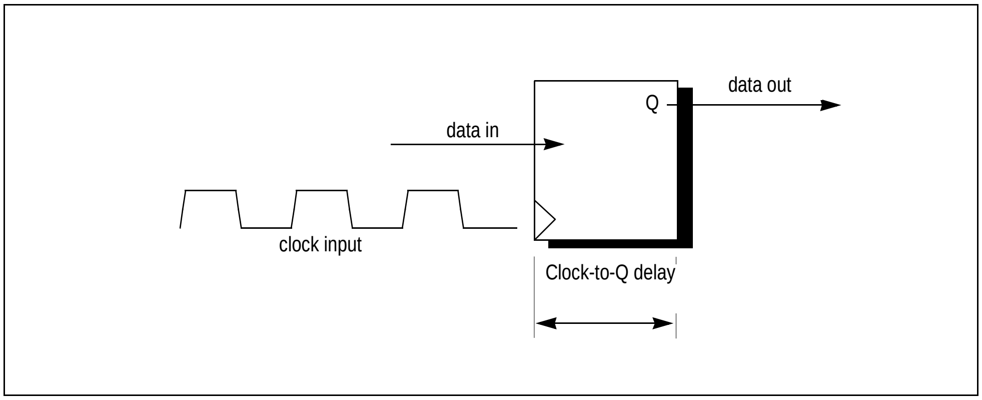


Figure 1.2 Clock-to-Q Delay

Notes

Performance Overview

The RC32364 brings RISCore4000^{TM} 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. These techniques combine to allow the system designer more than 1 GB/sec aggregate internal cache bandwidth, 300MB/sec external bus bandwidth, 175 Dhrystone MIPS.

An array of development tools as well as integrated in-circuit emulation support facilitates rapid development of RC32364-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 RISCore3000^{TM} family, the RC32364 will serve in many of the same applications. The RC32364 also supports applications that require integer digital signal processing (DSP) functions.

Features

- *High-performance embedded 32-bit RISCore32300^{TM} microprocessor*
  - *Based on MIPS-II RISC architecture with enhancements*
  - *Scalar 5-stage pipeline minimizes branch and load delays*
  - *DSP engine capable of doing 1 Mul_Acc instruction every 2 clock cycles*
- *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, achieves over l GB/sec bandwidth*
- *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*
- *Flexible bus interface allows simple, low-cost designs*
  - *Bus interface runs at a fraction of pipeline rate*
  - *Programmable port-width interface (8-,16-, 32-bit memory and I/O regions)*
  - *Programmable bus turnaround (BTA) times*
  - *Supports single datum or burst transactions*
  - *Selectable system byte-ordering*
- *Improved real-time support*
  - *Fast interrupt decode*
- *Low-power operation*
  - *Active power management: powers down inactive units*
  - *Active mode <1W, typical*
  - *Stand-by mode <20mW*
- *Enhanced JTAG interface, for low-cost In-circuit Emulation (ICE)*
- *MIPS architecture ensures applications software compatibility throughout the RISController series of embedded processors.*
- *Industrial temperature range support*
- *Windows® CE compatible*

Notes

Device Overview

The RC32364 has a level of integration designed for high-performance and high bandwidth computing. Key elements of the RC32364 are illustrated in the block diagram provided in Figure 1.3. An overview on these features follows, with more detailed explanations provided in subsequent chapters.

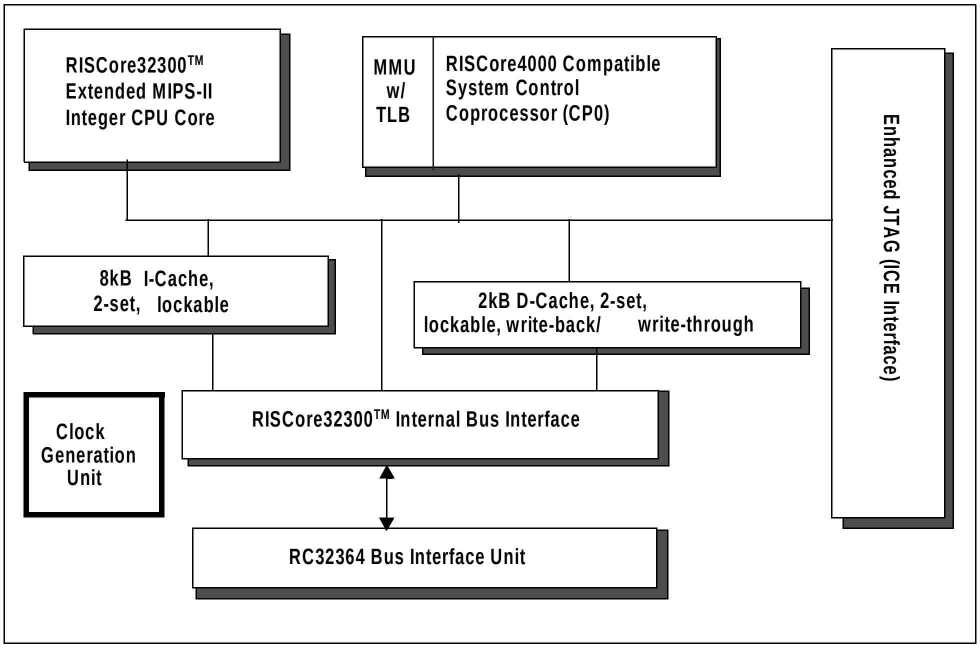

RISCore32300™ Extended MIPS-II Integer CPU Core
MMU w/ TLB
RISCore4000 Compatible System Control Coprocessor (CP0)
Enhanced JTAG (ICE Interface)
8kB I-Cache, 2-set, lockable
2kB D-Cache, 2-set, lockable, write-back/ write-through
Clock Generation Unit
RISCore32300™ Internal Bus Interface
RC32364 Bus Interface Unit

Figure 1.3 RC32364 Block Diagram

CPU Registers

The RC32364 has 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.

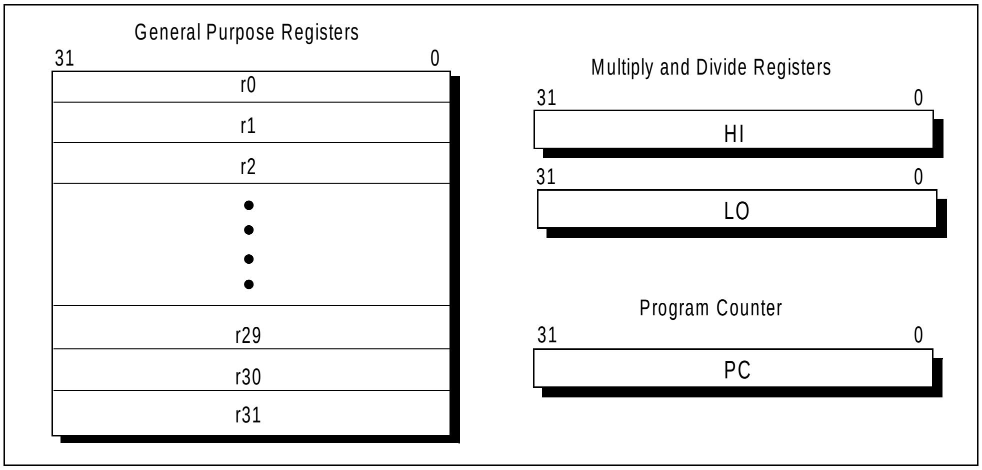


Figure 1.4 RC32364 CPU Registers

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

Notes

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 RISCore32300 processor 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 RISCore32300 implements the MIPS-II instruction set architecture (ISA) with enhancements that include:

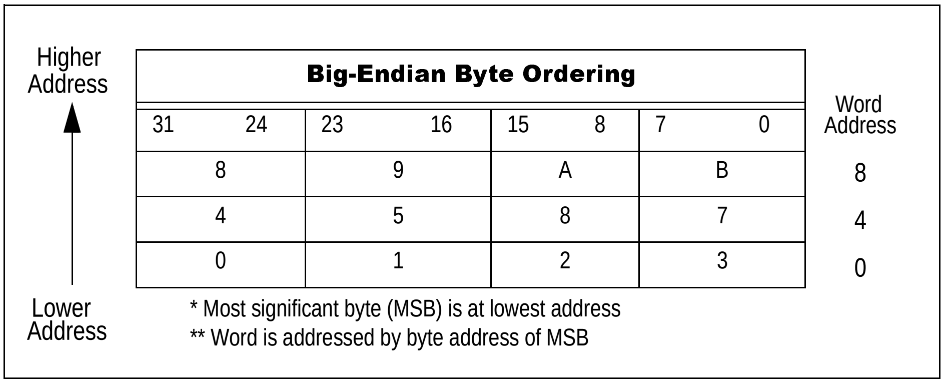
- *addition of MIPS-IV PREF operation, with various hint subfields*
- *addition of MIPS-IV conditional move instructions*
- *addition of MAD, MUL and MSUB instructions to the integer multiply units, used to perform multiply accumulate and multiply subtract operations*
- *addition of two new instructions: Count Leading Ones (CLO) and Count Leading Zeros (CLZ).*

These features come together to make the CPU well suited to DSP applications.

Configuration

During hardware reset, the RC32364's byte ordering is configurable into either a *big-endian* or *little-endian* convention. When con

d as a big-endian system, byte 0 is always the most significant (leftmost) byte in a word (see Figure 1.5). However, when configured as a little-endian system, byte 0 is always the least significant (rightmost) byte in a word (see Figure 1.6).

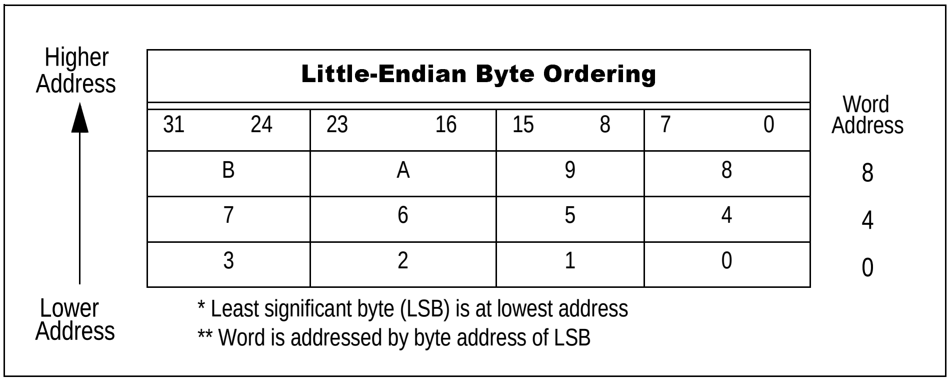


Big-Endian Byte Ordering

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 1.5 Big-Endian Byte Ordering Convention



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 1.6 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 RC32364 implements an RISCore4000 family compatible CP0. Specific details on the CP0 registers implemented by the RC32364 are provided in Chapter 5, “CPU Exception Processing.”

Memory Management Unit (MMU)

The RC32364’s MMU is modeled after the MMU found in the RISCore4000 family and includes the Translation Lookaside Buffer (TLB). This MMU offers the following advantages, relative to the traditional RISCore3000 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 RC32364 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 RC32364 supports a cache-locking feature, which is implemented on a “per-line basis,” enabling the system designer to maximize the system’s cache efficiency.

For more specific details on the characteristics and organization of these caches, refer to Chapter 6 of this manual, “Cache Organization, Operation and Coherency.”

Bus Interface Unit (BIU)

The RC32364 system interface is through a multiplexed data bus. The RC32364 implements special techniques for byte-enable generation and for low-order address bits in block refills. The data interface is 32-bits wide with a variable port width interface, including 8-bit boot-prom support.

The system interface unit is 32-bits wide and includes features such as multiple pipeline-to-system clock ratios support; DMA arbiter interface; variable port-width interface, including 8-bit boot prom; sub-block ordering read and sequential ordering write. To simplify design, the same data lines are used for a given width of memory, regardless of memory byte-ordering (endianness). A complete pin description table follows and is also provided in Chapter 9, “Bus Interface Overview.”

> **Note:** Although the internal cache is parity protected, parity for the external bus is not supported. Parity is generated into the cache on refill and checked when the data is brought to the execution core.

Notes

Pin Description Table

The following is a list of bus interface pins that are available on the RC32364. Pin names followed by an asterisk (*) are active when low.

<table>
<tr><th>Pin</th><th>Type</th><th>Description</th></tr>
<tr><td colspan="3"><b>System Interface</b></td></tr>
<tr><td>AD(31:4)</td><td>I/O</td><td>Addr(31:4)/Data(31:4)
High-order multiplexed address and data bits. Regardless of system byte ordering, AD(31) is the MSB of the address.</td></tr>
<tr><td>AD(3:0)</td><td>I/O</td><td>Size(3:0)/Data(3:0)
Valid sizes for the RC32364 are as follows:
<table>
<tr><th>Size(3)</th><th>Size(2)</th><th>Size(1)</th><th>Size(0)</th><th>Transfer Width</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>16 bytes</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1 byte</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2 bytes</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>3 bytes</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4 bytes</td></tr>
</table>
Other encodings allow future generations to service other transfer sizes. During the data phase, AD[3:0] represents the Data(3:0).</td></tr>
<tr><td>Addr(3:2)</td><td>O</td><td>Addr(3:2)
Non-multiplexed address lines. These serve as the word within block address for cache refills (Addr(3:2)). The word within block address bits count in a sub-block ordering.</td></tr>
<tr><td>ALE</td><td>O</td><td>Address Latch Enable
This signal provides set-up and hold times around the address phase of the AD bus.</td></tr>
<tr><td>ADS*</td><td>O</td><td>Address Strobe
This active-low signal indicates valid address and the start of a new bus transaction. The processor asserts ADS* for the entire address cycle. This is the inverse of the ALE signal.</td></tr>
<tr><td>Width(1:0)</td><td>O</td><td>Bus Width
Indicates the Physical Memory/IO data bus size as follows:
<table>
<tr><th>Width(1)</th><th>Width(0)</th><th>Port Width</th></tr>
<tr><td>0</td><td>0</td><td>8 bits</td></tr>
<tr><td>0</td><td>1</td><td>16 bits</td></tr>
<tr><td>1</td><td>0</td><td>32 bits</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>BE*(3:0)</td><td>O</td><td><b>ByteEnables(3:0)/Addr(1:0)</b>
Indicates which byte lanes are expected to participate in the transfer.
<table>
<tr><th></th><th colspan="4">Byte Lanes Enabled In Data Transfer</th></tr>
<tr><th>Port Width</th><th>BE(3)</th><th>BE(2)</th><th>BE(1)</th><th>BE(0)</th></tr>
<tr><td>32-bit</td><td>Used</td><td>Used</td><td>Used</td><td>Used</td></tr>
<tr><td>16-bit</td><td>Byte High Enable</td><td>Not Used</td><td>Address Bit 1 (A1)</td><td>Byte Low Enable</td></tr>
<tr><td>8-bit</td><td>Not Used (Driven High)</td><td>Not Used (Driven High)</td><td>Address Bit 1 (A1)</td><td>Address Bit 0 (A0)</td></tr>
</table></td></tr>
</table>

Table 1 System Interface Pin Descriptions (Part 1 of 3)

Notes

Pin	Type	Description
CIP*	O	Cycle-in-progress Denotes that a cycle is in progress. Asserted in the address phase and continue asserted until the ACK* for the last data is sampled.
I/D*	O	I/D* Indicates that the current cycle is for an instruction (active high) or data (active low) transaction.
Rd*	O	Read This active-low signal indicates that the current transaction is a read.
Wr*	O	Write This active-low signal indicates that the current cycle transaction is a write.
DataEn*	O	Data Enable This active-low signal indicates that the AD bus is in data cycle. DEN* is asserted after the address cycle (starting of data cycle), and deasserted at the end of the last data cycle.
DT/R*	O	Data Transmit/Receive This active-low signal indicates the current cycle transaction of data direction. “High” is for a write cycle and “Low” is for a read cycle.
Ack*	I	Acknowledge Receiving Data On read transactions, this signal indicates to the RC32364 that the memory system has placed valid data on the A/D bus, and that the processor may move the data into the on-chip Read Buffer. On a write transaction, this indicates to the RC32364 that the memory system has accepted the data on the A/D bus.
Last*	O	Last Data This active-low output is used to indicate the last data phase of a transfer.
Handshake Interface		
BusErr*	I	Bus Error Indicates that a bus error has occurred.
Retry*	I	Retry Indicates that the current bus cycle must be terminated. Retry* is ignored after acceptance of the first data during a read cycle. During a write, Retry* is recognized in all data cycles.
Initialization Interface		
ColdReset*	I	ColdReset This active-low signal is used for power-on reset.
Reset*	I	Reset This active-low signal is used for both power-on and warm reset.
DMA Interface		
BusReq*	I	Bus Request This active-low signal is an input to the processor that is used to request mastership of the external interface bus. Mastership is granted according to the assertion of this input, taken back based on its negation.
BusGnt*	I/O	Bus Grant/ModeBit(5) This active-low signal is an output from the processor and is used to indicate that the CPU has relinquished mastership of the external interface bus. BusGnt* goes low initially for at least 2 clocks to indicate that the CPU has relinquished mastership of the external interface bus. After going low, BusGnt* returns high, either when the CPU makes an internal request for the bus or after BusReq* is de-asserted.During the power-on reset (Cold Reset), BusGnt* is an input, ModeBit(5).
Interrupt Interface		
NMI*	I	Non-Maskable Interrupt NMI is falling edge sensitive and an asynchronous signal.
Int*(5:0)	I	Interrupt/ModeBit(9:6) These interrupt inputs are active low to the CPU. During power-on, Int*(3:0) serves as ModeBit(9:6).

Table 1 System Interface Pin Descriptions (Part 2 of 3)

Notes

Pin	Type	Description
Debug Emulator Interface		
Tclk	I	Testclock An input test clock, used to shift into or out of the Boundary-Scan register cells. Tclk is independent of the system and the processor clock with nominal 50% duty cycle.
TDI/DINT*	I	TDI/DINT* On the rising edge of Tclk, 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.
TDO/TPC	O	TDO/TPC The TDO is serial data shifted out from instruction or data register on the falling edge of Tclk. When no data is shifted out, the 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.
TMS	I	TMS The logic signal received at the TMS input is decoded by the TAP controller to control test operation. TMS is sampled on the rising edge of the TCLK. Requires an external pull-up on the board.
TRST*	I	TRST* The TRST* pin is an active-low signal for asynchronous reset of the debug unit, independent of the processor logic. Requires an external pull-down on the board.
DCLK	O	DCLK Processor Clock. During Real Time Mode, this signal is used to capture address and data from the TDO signal at the processor clock speed or any division of the internal pipeline.
PCST(2:0)	I/O	PCST(2:0)/ModeBit(2:0) 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 During power-on reset (cold reset), PCST(2:0) serves as ModeBit(2:0).
PCST(4:3)	I/O	PCST(4:3)/ModeBit(4:3) PC Trace Status Information. Reserved Pins for future expansion. During power-on reset, PCST(4:3) serves as ModeBit(4:3).
DebugBoot	I	DebugBoot The Debug Boot 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].
Clock/Control Interface		
MasterClk	I	MasterClock This input clock is the bus clock. The core frequency is derived by multiplying this clock up.
VccP	I	VccP Quiet Vcc for PLL.
VssP	I	VssP Quiet Vss for PLL.
Vcc I/O	I	Supply voltage for output buffers.
Vcc Core	I	Supply voltage for internal logic.
Vss	I	Ground.

Table 1 System Interface Pin Descriptions (Part 3 of 3)

Notes

Address/Data Interfaces

The RC32364 supports a 32-bit wide system data interface through a multiplexed address/data bus. The RC32364 implements special techniques for byte-enable generation and for low-order address bits in block refills. The data interface is 32-bits wide. There is no parity support for the bus, although the internal cache is parity protected. Parity is generated into the cache on refill and then checked when the data is brought to the execution core. The RC32364 supports a variable port-width interface.

Read Control Interface

The read control interface is designed to easily support a variety of read transactions:

- *a single data read that can be satisfied in one data phase*
- *a single data read that requires multiple data phases (i.e. data is wider than the port)*
- *a cache-block read to any of the valid port widths*
- *sub-block ordering accessing among the words*

Write Control Interface

Writes can be single data (including mini-bursts to narrow ports) or cache-line write-backs. For the case of cache line writebacks, data is written using sequential ordering, starting from word 0 of the line.

Bus Retry

External devices can force the RC32364 bus to terminate the current bus transaction by asserting Retry* (without asserting Ack*), during the first data phase. On a read operation, once the first data is accepted (by sampling the Ack*), Retry* will be ignored.

Bus Error

If the RC32364 tries to access a memory region that does not have any device connect to it or if external logic detects a condition such as a parity error, BusErr* can then be asserted, which forces RC32364 into taking the bus error exception. BusErr* is ignored after the acceptance of the first data. If BusErr* and Ack* are both sampled asserted in the first data phase, Ack* will be ignored, and the bus error exception will be taken.

DMA Mastership Interface

The DMA interface uses a simple two signal protocol to allow an external agent mastership of the system bus. Logic internal to the CPU synchronizes the external interface to the internal arbiter unit to insure that no conflicts occur between the internal synchronous requesters (read and write engines) and the external asynchronous (DMA) requester.

Clocking Interface

All bus timings are with respect to an external input clock, which is provided as the bus interface clock. An on-chip PLL multiplies this input clock by a factor selected at device reset, to provide the pipeline clock. Thus, MasterClk is the only relevant clock to a system designer.

Reset Interface

The RC32364 processor has two types of resets that use the ColdReset* and Reset* input signals: **Power-on reset** and **Warm reset.** During a Warm reset, the processor preserves the internal state and takes the reset exception. During the power-on reset sequence, the RC32364 obtains configuration information using its mode configuration interface.

The RC32364's initialization values are obtained from PCST[4:0], BusGnt and Int*[4:0] and are the same as ModeBit[9:0] during the power-on reset. The values of ModeBit[9:0] are latched with the ColdReset* signal.

Notes

Power Reduction Modes

The RISCore32300 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 (Int[5:0]*, NMI*, Reset*, and ColdReset*) will continue to run. If the bus is not idle when the WAIT instruction finishes the W pipe-stage, the WAIT is treated as a NOP. Once the CPU is in Standby Mode, any interrupt, including the internally generated timer interrupt, will cause the CPU to exit Standby Mode.

CPU Instruction Set Overview

Notes

Introduction

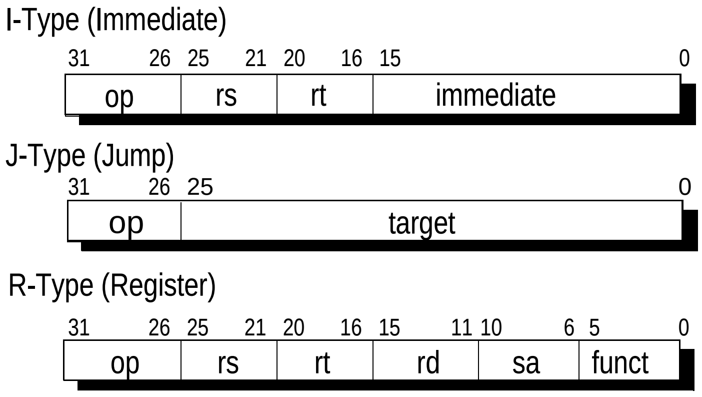
This chapter provides a general overview on the three CPU instruction set formats of the MIPS architecture: Immediate, Jump, and Register. For more details on a specific CPU core instruction, refer to the *IDT MIPS Microprocessor Family Software Reference Manual*. For descriptions of the new instruction sets implemented in this device, refer to Appendix A of this manual.

CPU Instruction Formats

Each CPU instruction consists of a single 32-bit opcode, aligned on a word (4-byte) boundary. As shown in Figure 2.1, 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 2.1 CPU Instruction Formats

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 Reference Manual*.

Notes

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 RC32364 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 RC32364 processor data item to be loaded or stored, set by the load or store instruction opcode. Access types are defined in Appendix A of the *IDT MIPS Microprocessor Family Software Reference Manual*.

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 2.1. 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	▨	▨	▨

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, RC32364 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 2.1 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 Reference Manual*.

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 2.1 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.

Notes

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.

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 Appendix A of the *IDT MIPS Microprocessor Family Software Reference Manual*.

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 Appendix A of the *IDT MIPS Microprocessor Family Software Reference Manual*.

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.

Notes

Summary of CPU Supported Instruction Sets

The tables that follow list instructions supported by the RC32364 core. Load and Store Instructions are listed in Table 2.2, Arithmetic Instructions (ALU Immediate) in Table 2.3, Arithmetic Instructions (3-Operand, R-Type) in Table 2.4, Multiply, Divide and DSP Instructions are in Table 2.5, Jump and Branch Instructions are in Table 2.6, Shift Instructions are in Table 2.7, Coprocessor Instructions are in Table 2.8, Special Instructions are listed in Table 2.9, Exception and CP0 Instructions are listed in Table 2.10 and Table 2.11.

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
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 2.2 Load and Store Instructions

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 2.3 Arithmetic Instructions (ALU Immediate)

Opcode	Description	MIPS ISA Level
ADD	Add	I
ADDU	Add Unsigned	I
SUB	Subtract	I

Table 2.4 Arithmetic Instructions (3-Operand, R-Type) (Part 1 of 2)

Notes

Opcode	Description	MIPS ISA Level
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 2.4 Arithmetic Instructions (3-Operand, R-Type) (Part 2 of 2)

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	RC4650, RISCore32300, RC64574/575
MAD	Multiply Add	RC4650, RISCore32300, RC64574/575
MADU	Multiply Add Unsigned	RC4650, RISCore32300, RC64574/575
MSUB	Multiply Subtract	RC32364, RC64574/575
MSUBU	Multiply Subtract Unsigned	RC32364, RC64574/575
CLZ	Count Leading Zeros	RC32364, RC64574/575
CLO	Count Leading Ones	RC32364, RC64574/575

Table 2.5 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

Table 2.6 Jump and Branch Instructions (Part 1 of 2)

Notes

Opcode	Description	MIPS ISA Level
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
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 2.6 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 2.7 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 2.8 Coprocessor Instructions

Notes

Opcode	Description	MIPS ISA Level
SYSCALL	System Call	I
BREAK	Break	I
DRET	Debug Exception Return	RC32364, RISCore32300
SDBBP	Software Debug Breakpoint	RC32364, RISCore32300

Table 2.9 Special Instructions

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 2.10 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	RISCore4000, RISCore32300
ERET	Exception Return	RISCore4000, RISCore32300
WAIT	Enter Standby mode	RISCore4000, RISCore32300

Table 2.11 CP0 Instructions

CPU Pipeline Architecture

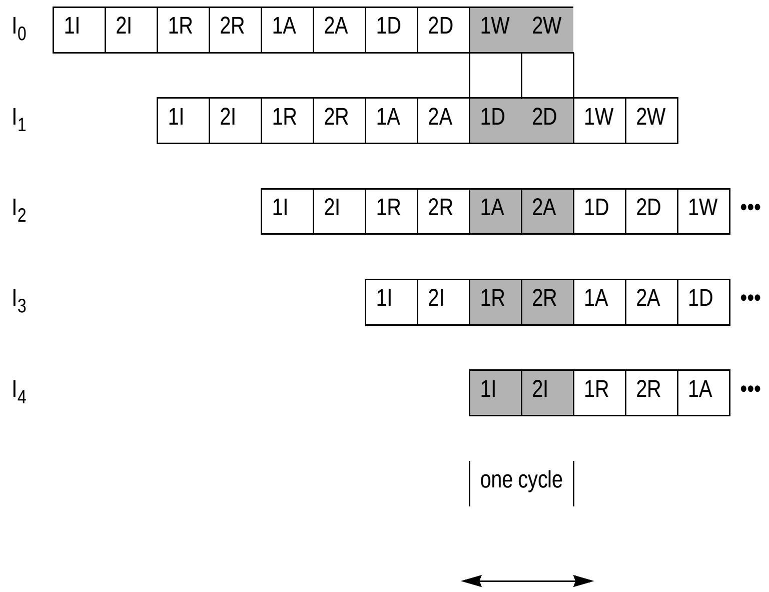
Notes

Introduction

The RISCore32300 uses a 5-stage instruction pipeline, similar to the RISCore3000 and RISCore4000 families. The simplicity of this pipeline enables the processor to achieve high frequency while minimizing device complexity.

The RISCore32300 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 RC32364 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 3.1.


I_0 1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
I_1 1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
I_2 1I 2I 1R 2R 1A 2A 1D 2D 1W •••
I_3 1I 2I 1R 2R 1A 2A 1D •••
I_4 1I 2I 1R 2R 1A •••
one cycle

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 3.1 Instruction Pipeline Stages

Notes

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*
- *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.*

Notes

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.*

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.

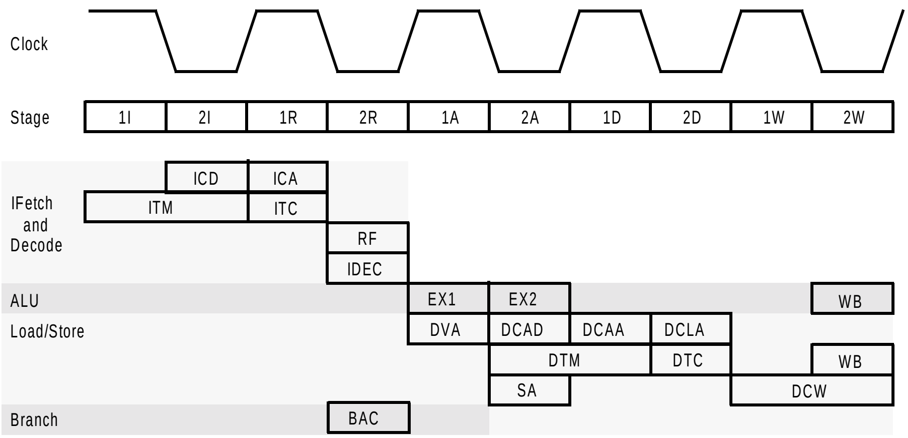
1W - Write Back, Phase One

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 3.2 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 3.2 Pipeline Activities

Notes

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.

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 3.3 illustrates the branch delay.

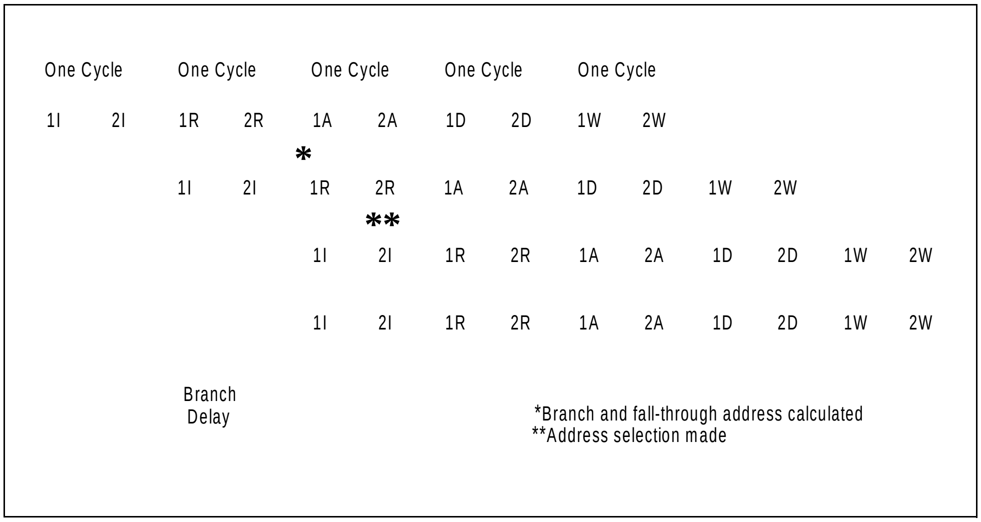

One Cycle
One Cycle
One Cycle
One Cycle
One Cycle
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
*
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
**
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Branch Delay
*Branch and fall-through address calculated
**Address selection made

Figure 3.3 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 3.4 shows the load delay of one pipeline cycle.

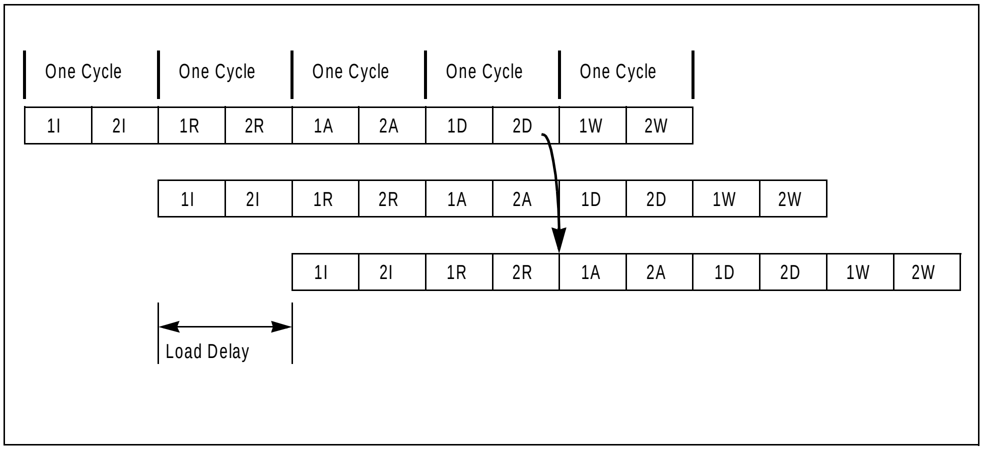

One Cycle
One Cycle
One Cycle
One Cycle
One Cycle
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Load Delay

Figure 3.4 CPU Pipeline Load Delay

Notes

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.

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 RC32364 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 3.5 shows the exception detection procedure (for example, a reserved instruction exception).

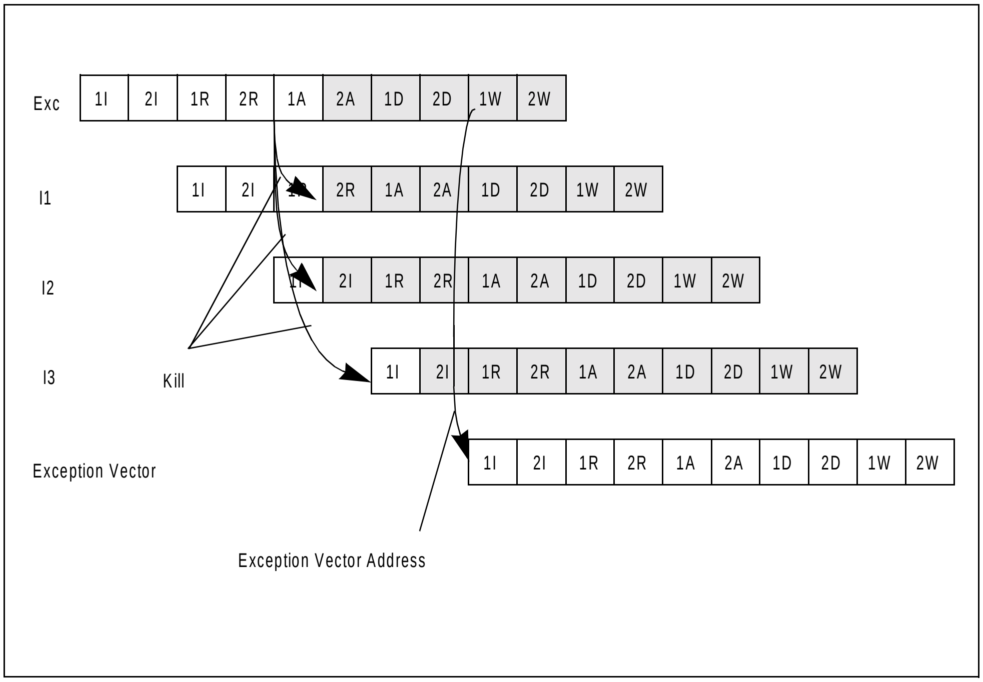
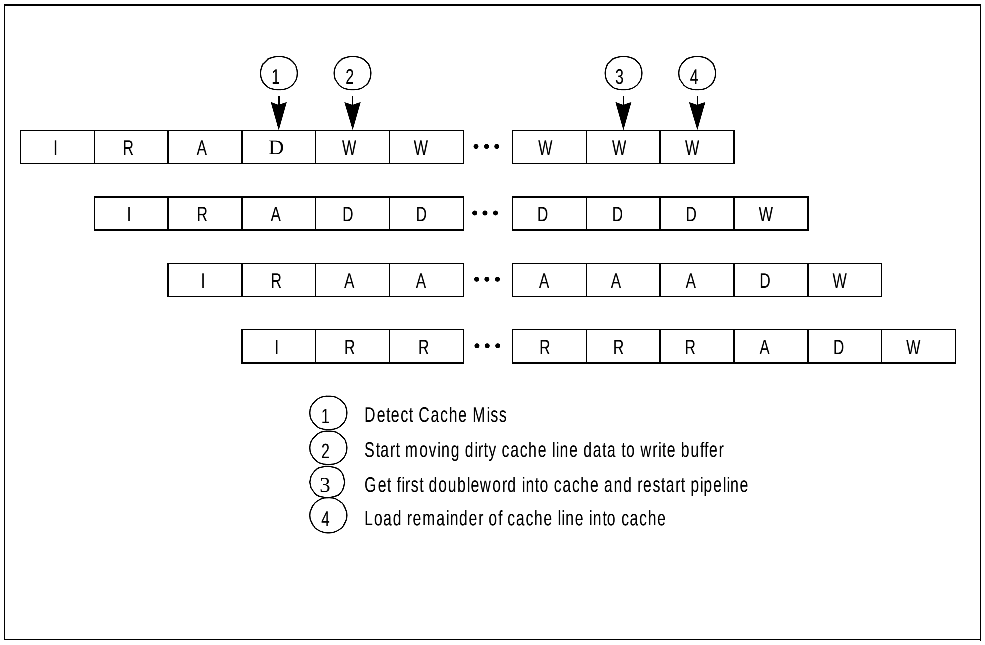

Exc
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
I1
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
I2
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
I3
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Kill
Exception Vector
1I 2I 1R 2R 1A 2A 1D 2D 1W 2W
Exception Vector Address

Figure 3.5 Exception Detection

Notes

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 3.6 illustrates a data cache miss stall.



1 Detect Cache Miss
2 Start moving dirty cache line data to write buffer
3 Get first doubleword into cache and restart pipeline
4 Load remainder of cache line into cache

Figure 3.6 Data Cache Miss

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.

Notes

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 3.7 shows an instruction cache miss.

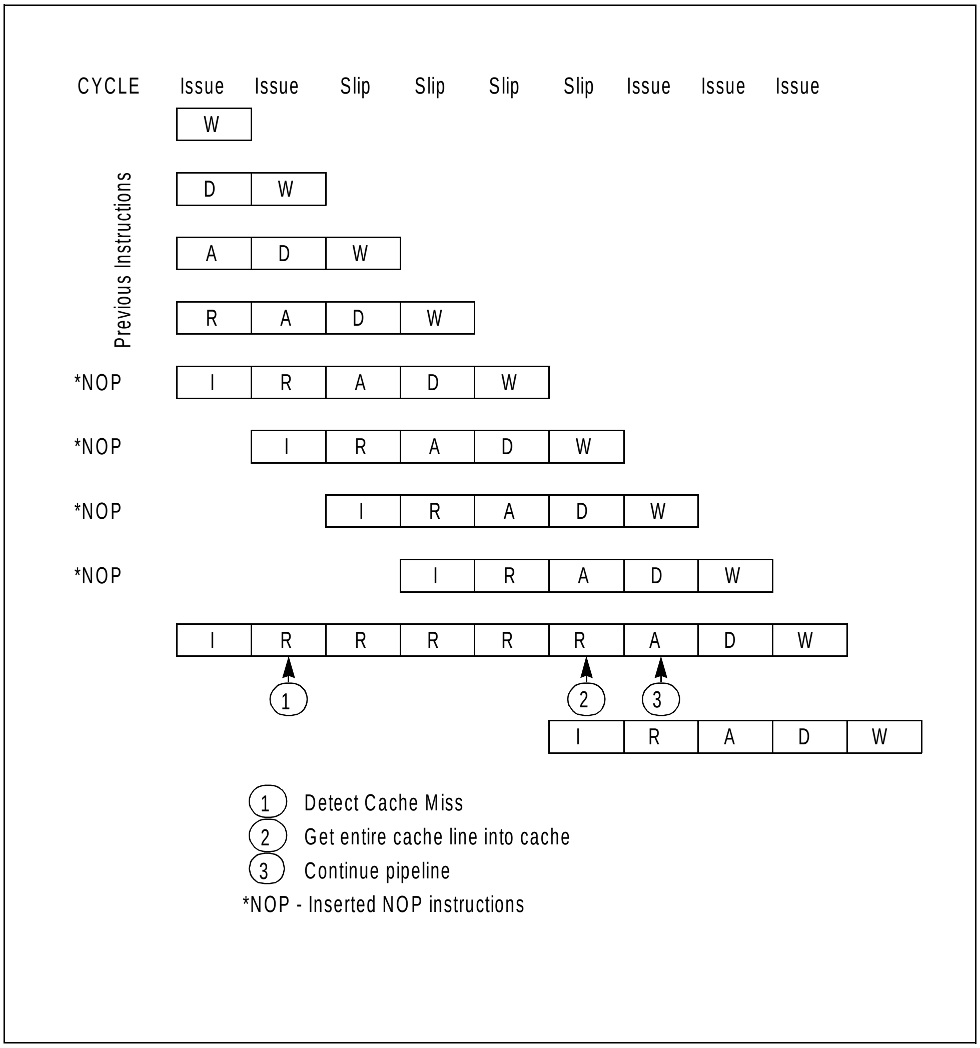


Figure 3.7 Instruction Cache Miss

As shown in Figure 3.7, 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.

Notes

Memory Management

Notes

Introduction

The Memory Management Unit (MMU) of the RC32364 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 4.1 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.

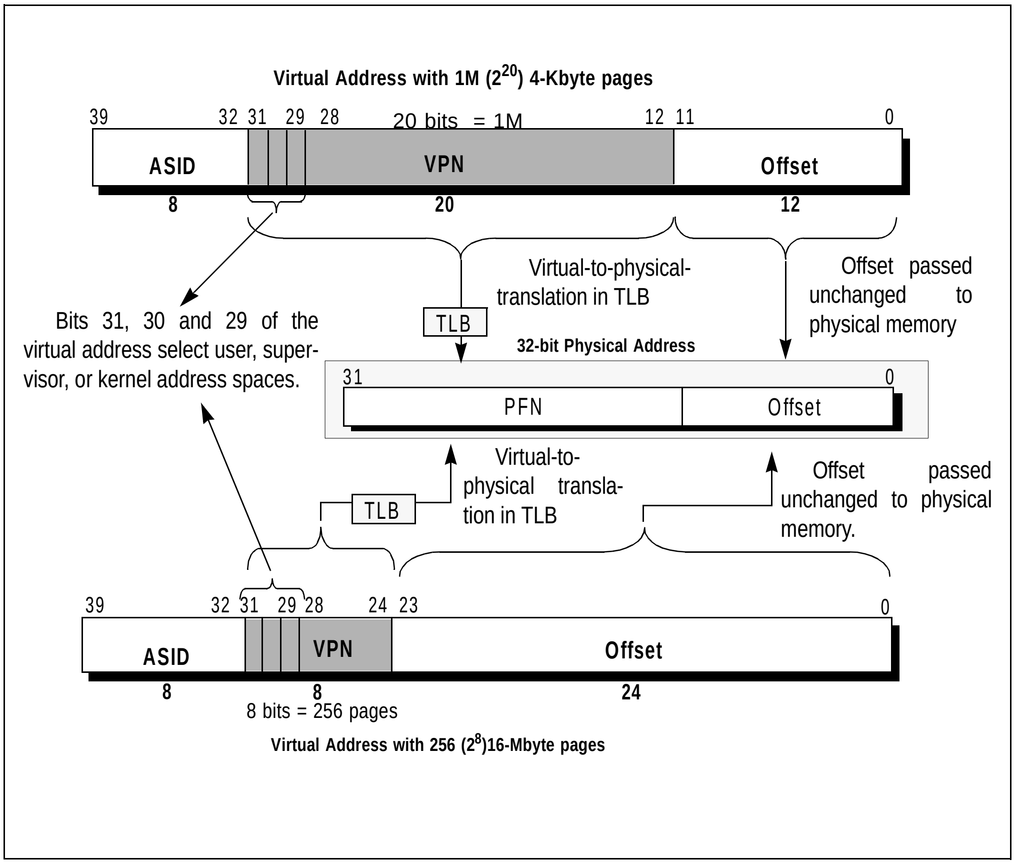

Virtual Address with 1M (2^{20}) 4-Kbyte pages
39
32
31
29
28
20 bits = 1M
12
11
0
ASID
VPN
Offset
8
20
12
Virtual-to-physical-translation in TLB
Offset passed unchanged to physical memory
Bits 31, 30 and 29 of the virtual address select user, supervisor, or kernel address spaces.
TLB
32-bit Physical Address
31
0
PFN
Offset
Virtual-to-physical translation in TLB
Offset passed unchanged to physical memory.
TLB
39
32
31
29
28
24
23
0
ASID
VPN
Offset
8
8
24
8 bits = 256 pages
Virtual Address with 256 (2^8)16-Mbyte pages

Figure 4.1 Overview of a 32-bit Virtual Address Translation

TLB Management

For fast virtual-to-physical address decoding, the RC32364 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 RC32364'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.

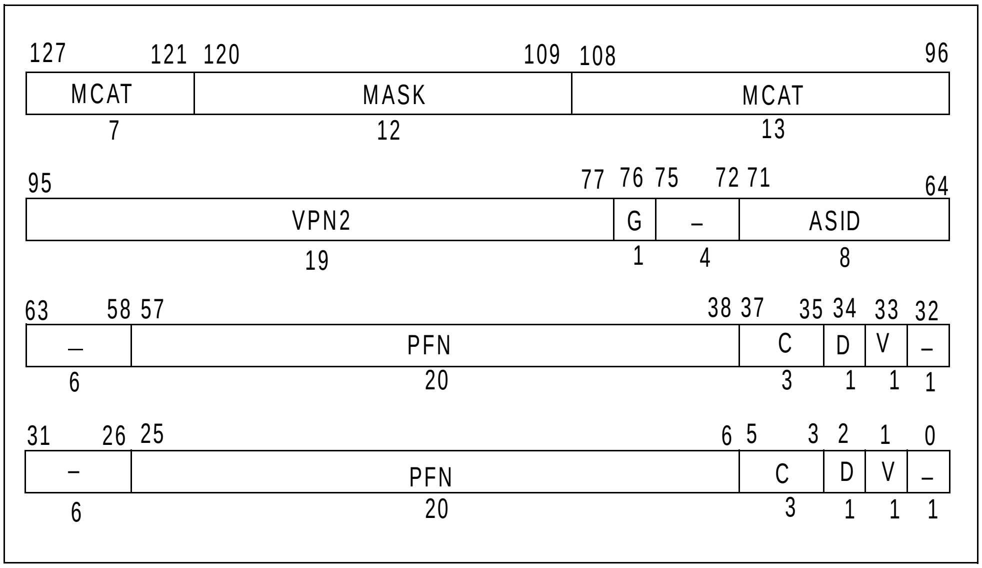


Figure 4.2 TLB Register Format

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>MASK</td><td>Page comparison mask</td></tr>
<tr><td>VPN2</td><td>Virtual Page Number divided by two (maps to two pages)</td></tr>
<tr><td>ASID</td><td>Address Space ID</td></tr>
<tr><td>G</td><td>Global. If this bit set, then ignore the ASID</td></tr>
<tr><td>PFN</td><td>Page Frame Number. Upper bit of physical address</td></tr>
<tr><td>C</td><td>Specifies the Cache Algorithm to be used, as shown below:
<table>
<tr><th><i>C</i> Value</th><th>Page Coherency Attribute</th></tr>
<tr><td>0</td><td>Cacheable, noncoherent, write-through, no write allocate</td></tr>
<tr><td>1</td><td>Cacheable, noncoherent, write-through, write allocate</td></tr>
<tr><td>2</td><td>Uncached</td></tr>
<tr><td>3</td><td>Cacheable, noncoherent, write-back</td></tr>
<tr><td>4:7</td><td>Reserved</td></tr>
</table>
</td></tr>
<tr><td>D</td><td>Dirty bit. This bit serves as a “write protect” bit</td></tr>
<tr><td>V</td><td>Valid bit. It set, TLB is valid. Otherwise a TLB Miss occurs</td></tr>
<tr><td>MCAT</td><td>Memory Controller Attributes. Reserved in RC32364 and must be written as ‘0’.</td></tr>
</table>

Table 4.1 TLB Register Field Descriptions

Notes

MMU Register Descriptions

The CP0 registers required to implement the RC32364 memory management unit are listed in Table 4.2. 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 4.2 RC32364 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 RC32364 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.

RC32364 implements additional bits in anticipation of derivative products. Figure 4.3 shows the format of the *Index* register; Table 4.3, which follows the figure, describes the contents of the *Index* register fields.

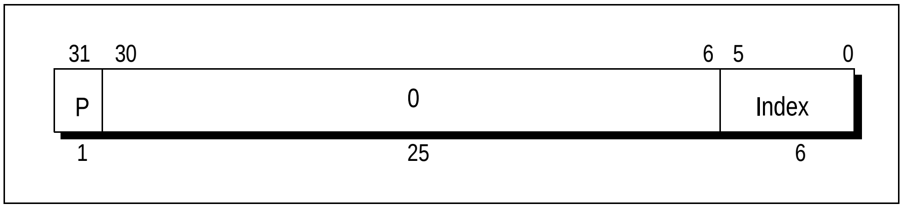


Figure 4.3 Index Register Format

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 4.3 Index Register Field Descriptions

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 RC32364 implements this register differently from the RC4000 family of processors. The R4C000 counts both valid and invalid instructions, the RC32364 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 4.4 shows the format of the *Random* register. Table 4.4 describes the contents of the *Random* register fields.

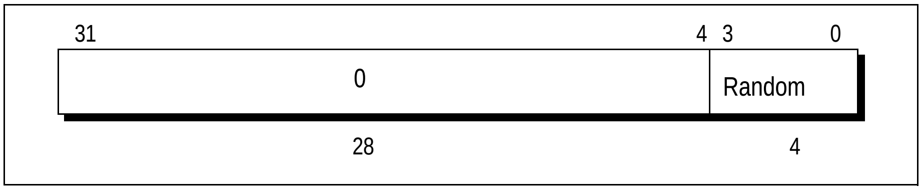


Figure 4.4 Random Register Format

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

Table 4.4 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 4.5 shows the format of this register. Table 4.5 provides descriptions for the fields of this register.

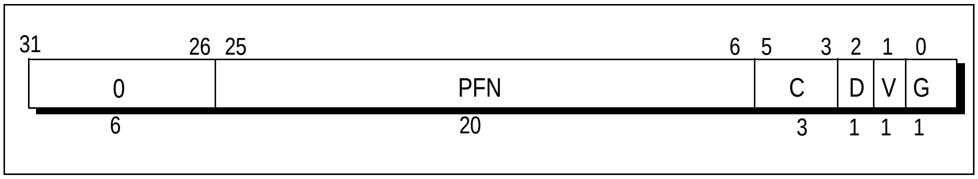


Figure 4.5 EntryLo0 and EntryLo1 Register Formats

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 4.5 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 4.6 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 4.6 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 4.6 illustrates the format of the *Context* register. Table 4.7 provides the descriptions of the *Context* register fields.

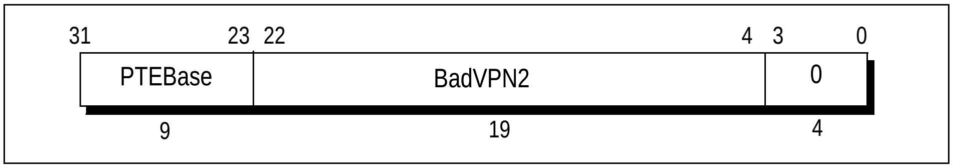


Figure 4.6 Context Register Format

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 4.7 Context Register Field Descriptions

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

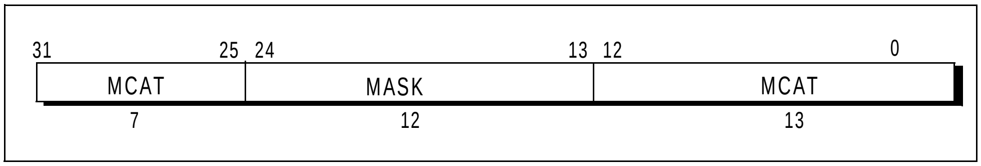


Figure 4.7 PageMask Register Format

Field	Description
Mask	Page comparison mask
MCAT	Memory controller attributes.

Table 4.8 PageMask Register Field Descriptions

Note: For the RC32364 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 4.8. “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.

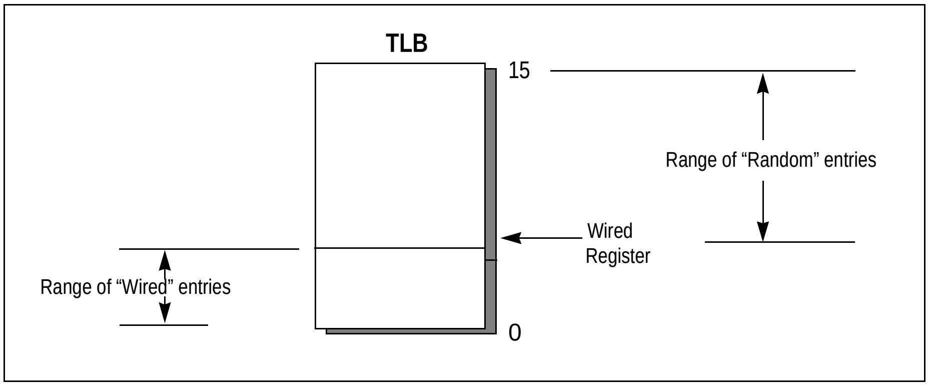


Figure 4.8 Diagram Showing Ranges of Wired and Random Entries

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 4.4 and Table 4.4). Figure 4.9 shows the format of the *Wired* register, and Table 4.9 lists the contents of this register’s fields.

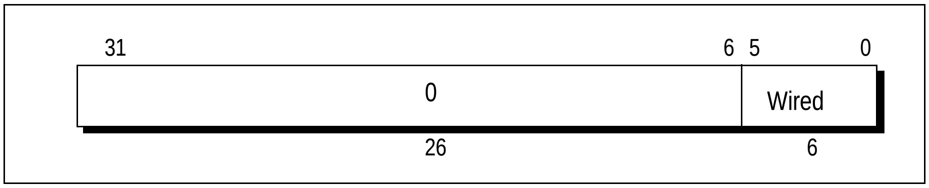


Figure 4.9 Wired Register Format

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 4.9 Wired Register Field Descriptions

Note that the RC32364 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 device, the RC32364 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 4.10 shows the format of the *BadVAddr* register.

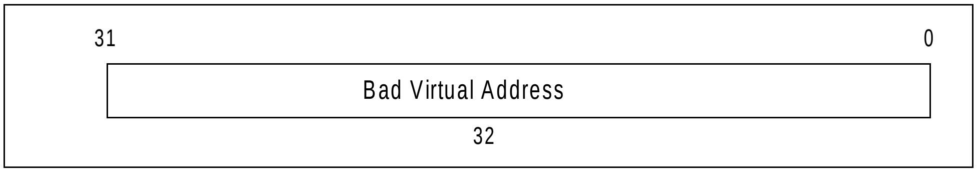


Figure 4.10 Bad Virtual Address Register (BadVAddr) Format

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 4.10 shows the Entry Hi register format and lists the field content descriptions.

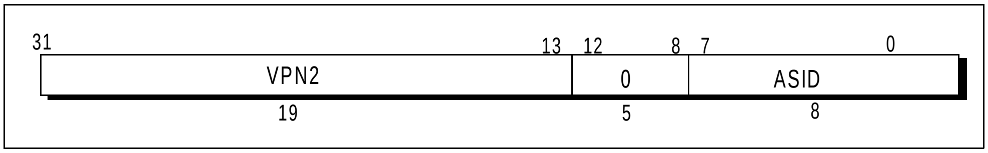


Figure 4.11 EntryHi Register Format

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 4.10 EntryHi Register Field Content Descriptions

Notes

Kernel/User Operating Modes and Addressing

The RC32364 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.

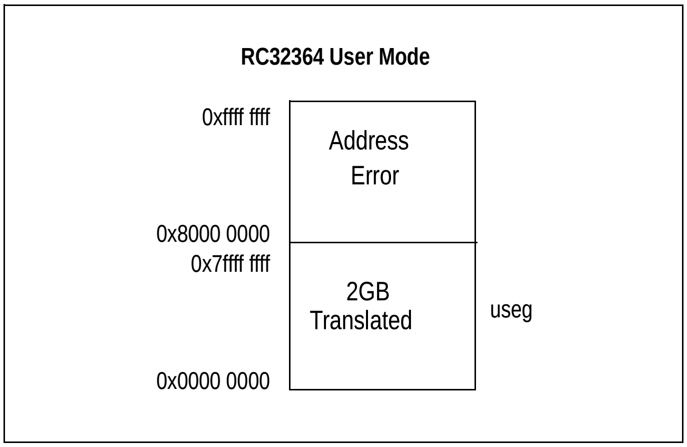


Figure 4.12 Illustration of RC32364 User Mode Address Space

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.

Notes

- *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, and the on-chip registers cannot be accessed.

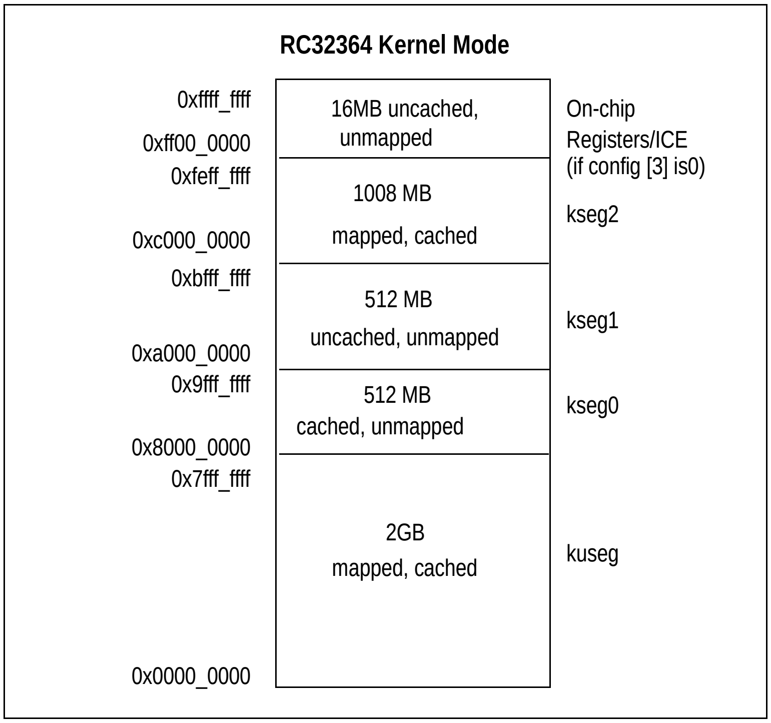


Figure 4.13 Illustration of RC32364 Kernel Mode Address Space

For complete field and content descriptions as well as virtual address locations for the Port Width and Bus Turnaround control registers, refer to Chapter 9 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 4). 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 RC32364 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*

Notes

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 5.1. Following the table, a brief operational description of each exception register is provided. Those listed as MMU registers are discussed further in Chapter 4, "Memory Management."

Number	Register	Description
0 - 8	___	Used for MMU registers. (See Chapter 4 for register descriptions)
9	Count	Timer Count
10	___	Used for MMU. (See Chapter 4 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
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 5.1 Basic CP0 Registers

Notes

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 5.1 shows the format of the *Count* register.

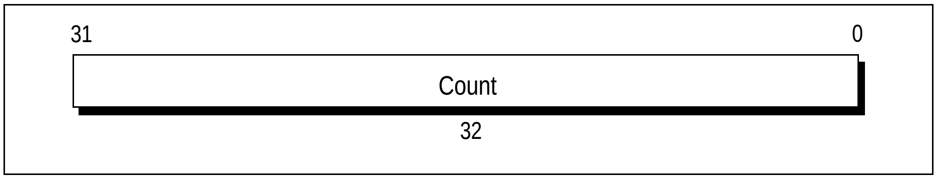


Figure 5.1 Count Register Format

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 5.2.

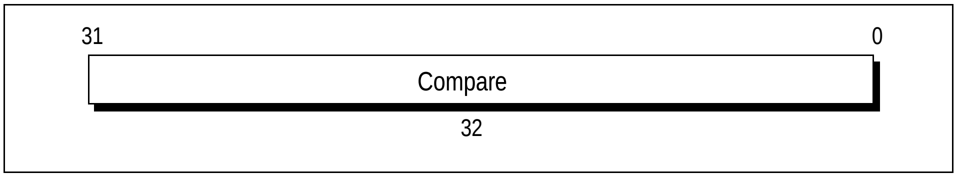


Figure 5.2 Compare Register Format

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 5.3 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.*

Notes

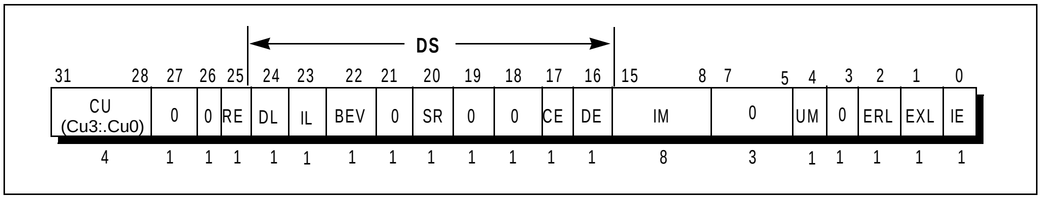


Figure 5.3 Status Register Format

Table 5.2 lists the descriptions for the Status register's fields.

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 5.2 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
NB = 0

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

Notes

Cause Register (13)

The 32-bit read/write *Cause* register describes the cause of the most recent exception. Figure 5.4 shows the fields of this register, and Table 5.3 describes the contents of the *Cause* register fields. As listed in Table 5.3, 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.

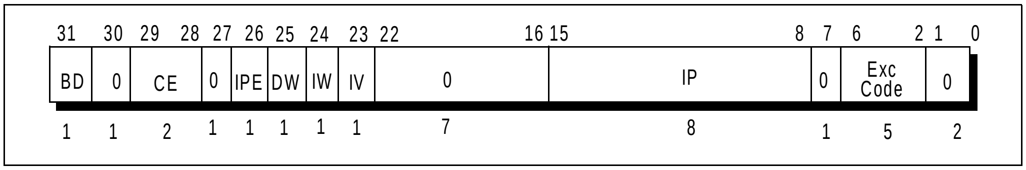


Figure 5.4 Cause Register Format

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.
IPE	Indicates the last exception is imprecise. 1 → imprecise exception 0 → precise exception
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 5.3 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

Table 5.4 Cause Register ExcCode Field (Part 1 of 2)

Exception Code Value	Mnemonic	Description
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 5.4 Cause Register ExcCode Field (Part 2 of 2)

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 5.5 shows the format of the *EPC* register.

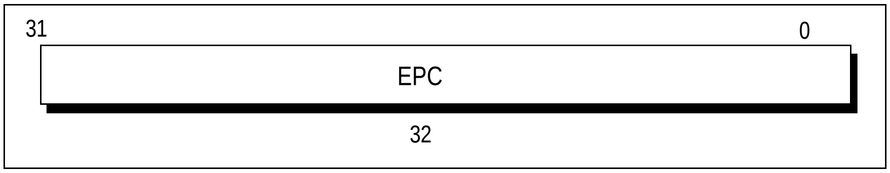


Figure 5.5 EPC Register Format

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 5.6 illustrates the format of the *PRId* register.

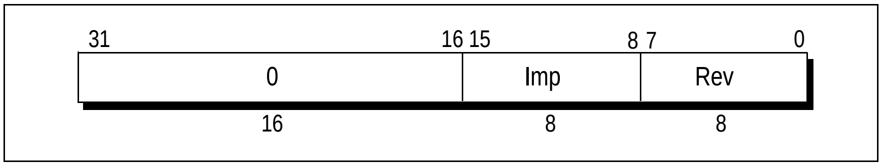


Figure 5.6 PRId Register Format

Table 5.5 describes the contents of the *PRId* register fields.

Field	Description
Imp	Implementation number RC32364: Imp = 26_{16}
Rev	Revision number
0	Reserved. Must be written as zeroes, returns zeroes when read.

Table 5.5 PRid Register Field Descriptions

Notes

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 RC32364 processor is 26_{16}. 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.

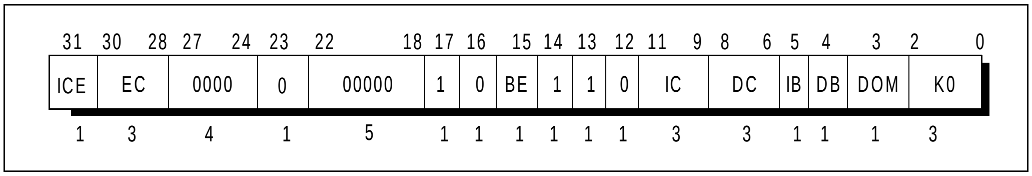
The revision number can distinguish some chip revisions; however, there is no guarantee that changes to the chip will be reflected in the *PRId* register, or that changes to the revision number necessarily reflects software-visible chip changes. For this reason, these values are not listed and software 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.

Config Register (16)

The *Config* register specifies various configuration options selected on the RC32364 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 5.7 shows the format of the *Config* register.



31	30 28	27 24	23	22 18	17	16	15	14	13	12	11 9	8 6	5	4	3	2 0
ICE	EC	0000	0	00000	1	0	BE	1	1	0	IC	DC	IB	DB	DOM	K0
1	3	4	1	5	1	1	1	1	1	1	3	3	1	1	1	3

Figure 5.7 Config Register Format

Table 5.6 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 RC32364 processor, this is set to 8 Kbytes (IC = 4)
DC	Primary D-cache Size (D-cache size = 2^{9+DC} bytes). In the RC32364 processor, this is set to 2 Kbytes (DC = 2)
IB	Primary I-cache line size 0 → 16 bytes (4 Words)

Table 5.6 Config Register Field Content Descriptions (Part 1 of 2)

Field	Description
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 4, “Memory Management”.)
Others	Reserved. Returns indicated values when read. Should be written with indicated values.

Table 5.6 Config Register Field Content Descriptions (Part 2 of 2)

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 5.8 shows the format of the *IWatch* register.

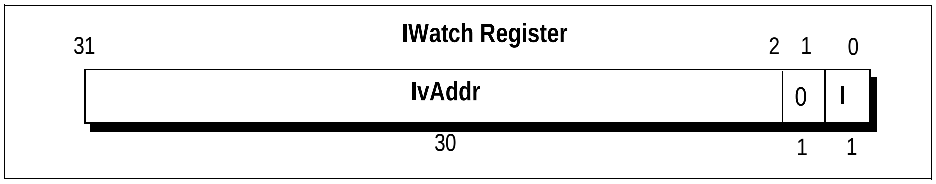


Figure 5.8 IWatch Register Format

Table 5.7 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 5.7 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 5.9.

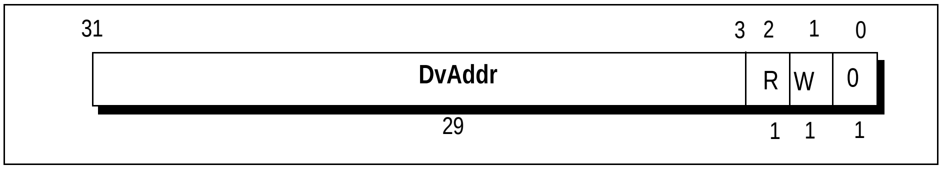


Figure 5.9 DWatch Register Format

Notes

Table 5.8 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 5.8 DWatch Register Field Descriptions

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 5.10 shows the format of the *ECC* register.

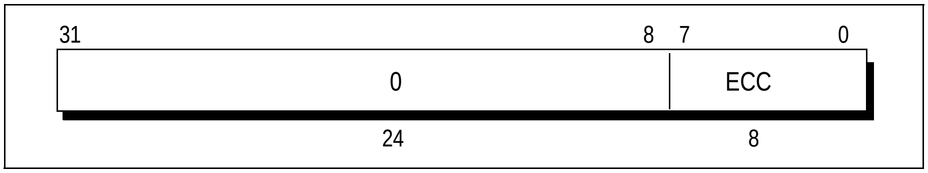


Figure 5.10 ECC Register Format

Table 5.9 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 5.9 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 5.11 shows the format of the *CacheErr* register.

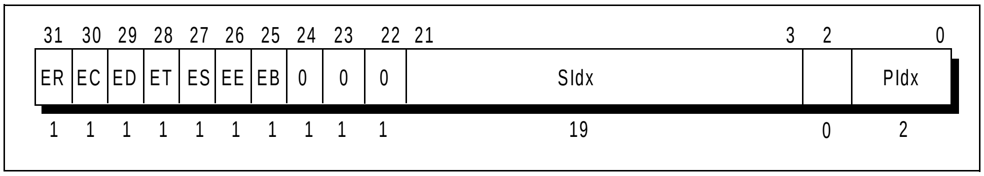


Figure 5.11 CacheErr Register

Table 5.10 provides descriptions on the contents of the *CacheErr* register fields.

Field	Description
ER	Indicates the type of reference as follows: 0 → instruction 1 → data
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 5.10 Cache Error Register Field Descriptions

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

Notes

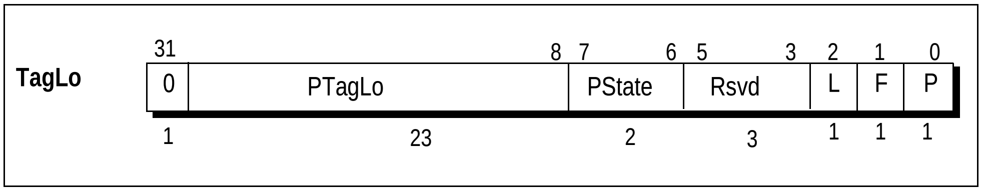


Figure 5.12 TagLo Register Format

Table 5.11 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 5.11 TagLo Register Field Descriptions

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

Table 5.12 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 5.13 shows the format of the *ErrorEPC* register.

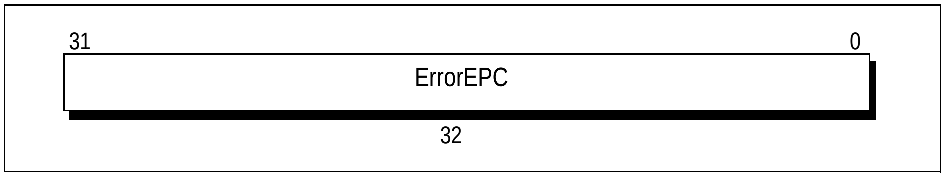


Figure 5.13 ErrorEPC Register

Notes

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).

General Exception Process

Figure 5.14 shows the process used for exceptions other than Reset, Soft Reset, NMI, and Cache Error.

```
T:  Cause ¨ BD || 0 || CE || $0^{12}$ || $\text{Cause}_{15:8}$ || 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 5.14 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 5.13.

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

Table 5.13 Exception Priority Order (highest to lowest) (Part 1 of 2)

Exception Priority			
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 5.13 Exception Priority Order (highest to lowest) (Part 2 of 2)

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 *kseg0*.

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 5.16. Table 5.14 lists the vector offset that is added to the base address to create the exception address.

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

Table 5.14 Base Address Vector Offset

As shown in Table 5.15, 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	RC32364 Processor Vector Offset
TLB refill, EXL = 0	0x000
Cache Error	0x100
Interrupt[1]	0x200
Others	0x180

Table 5.15 List of RC32364 Exception vectors

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

Exception	BEV	EXL	IV	ICE	RC32364 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 5.16 RC32364 Exception Vectors

Reset Exception

Cause: The Reset exception occurs when the **ColdReset***[1] 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

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 5.15.

[1] In the following sections (and throughout this manual) a signal reference that is followed by an asterisk (for example, **Reset***) is active when low.

```
T:  undefined
    Random ¨ TLBENTRIES–1
    Wired ¨ 0
    Config <- ICE || EC || EP || 00000000 || BE || 110 || 100 || 010 || 0 || 0 || 0 || 000
    ErrorEPC ¨ PC
    SR ¨ SR$_{31:23}$ || 1 || 0 || 0 || SR$_{19:3}$ || 1 || SR$_{1:0}$ / * ERL¨ 1, BEV ¨ 1 * /
    PC ¨ 0xBFC0 0000
```

Figure 5.15 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 Reset 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, 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. Soft Reset preserves the state of the caches and memory system, while resetting the bus state and cache state machine.

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 5.16.

```
T:  ErrorEPC ¨ PC
    SR ¨ $SR_{31:23}$ || 1 || 0 || 1 || $SR_{19:3}$ || 1 || $SR_{1:0}$ /* BEV¨ 1, SR ¨ 1, ERL ¨ 1 */
    PC ¨ 0xBFC0 0000
```

Figure 5.16 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*

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.

Notes

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.

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.

Notes

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.*

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 5.17.

Notes

```
T:  ErrorEPC ¨ PC
    CacheErr ¨ ER || EC || ED || ET || ES || EE || EB || $0^{25}$
    SR ¨ $SR_{31:3}$ || 1 ||$SR_{1:0}$                          /* Set ERL */
    if $SR_{22}$ = 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 5.17 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.

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.

[1] See Appendix A for instruction description.

Notes

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

Maskable: No.

Notes

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 RC32364 ISA 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 RC32364 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).

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 *Int-Mask* 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 5.8 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 5.7 for settings and descriptions.

Exception Handling and Servicing Flowcharts

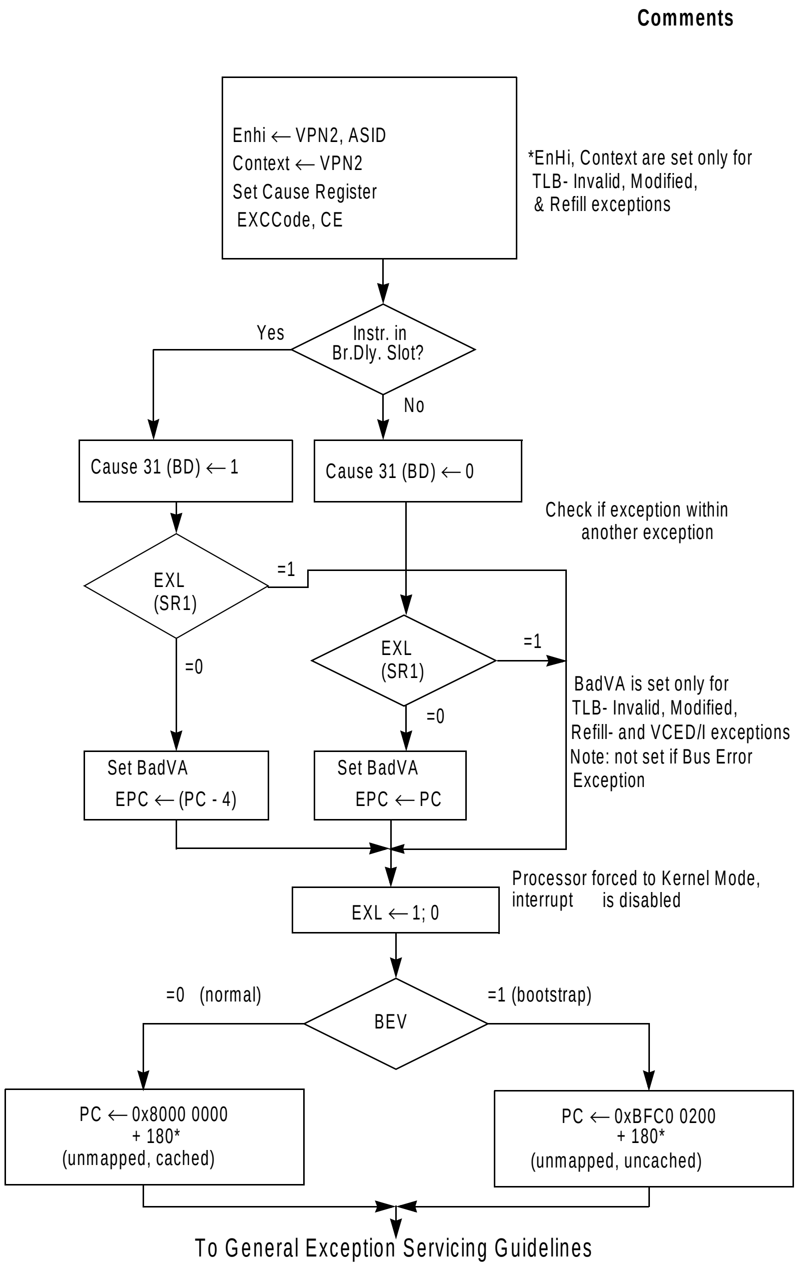
The remainder of this chapter contains flowcharts for the exceptions described in Table 5.13 as well as guidelines for their handlers.

Figure	Description
Figure 5.18 Figure 5.19	General exceptions and their exception handler (HW) General exceptions and their exception handler (SW)
Figure 5.20 Figure 5.21	TLB miss exception and their exception handler (HW) TLB refill exception servicing guideline (SW)
Figure 5.22	Cache error exception and its handler
Figure 5.23	Reset, soft reset and NMI exceptions, and a guideline to their handler.

Table 5.17 List of Exception Handling Flowchart Types

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

Notes



Exceptions other than Reset, Soft Reset, NMI, CacheErr or first-level TLB miss

Note: Interrupts can be masked by IE or IMs

* 200 for interrupts if Cause.IV is set.

Figure 5.18 General Exception Handling (HW)

Notes

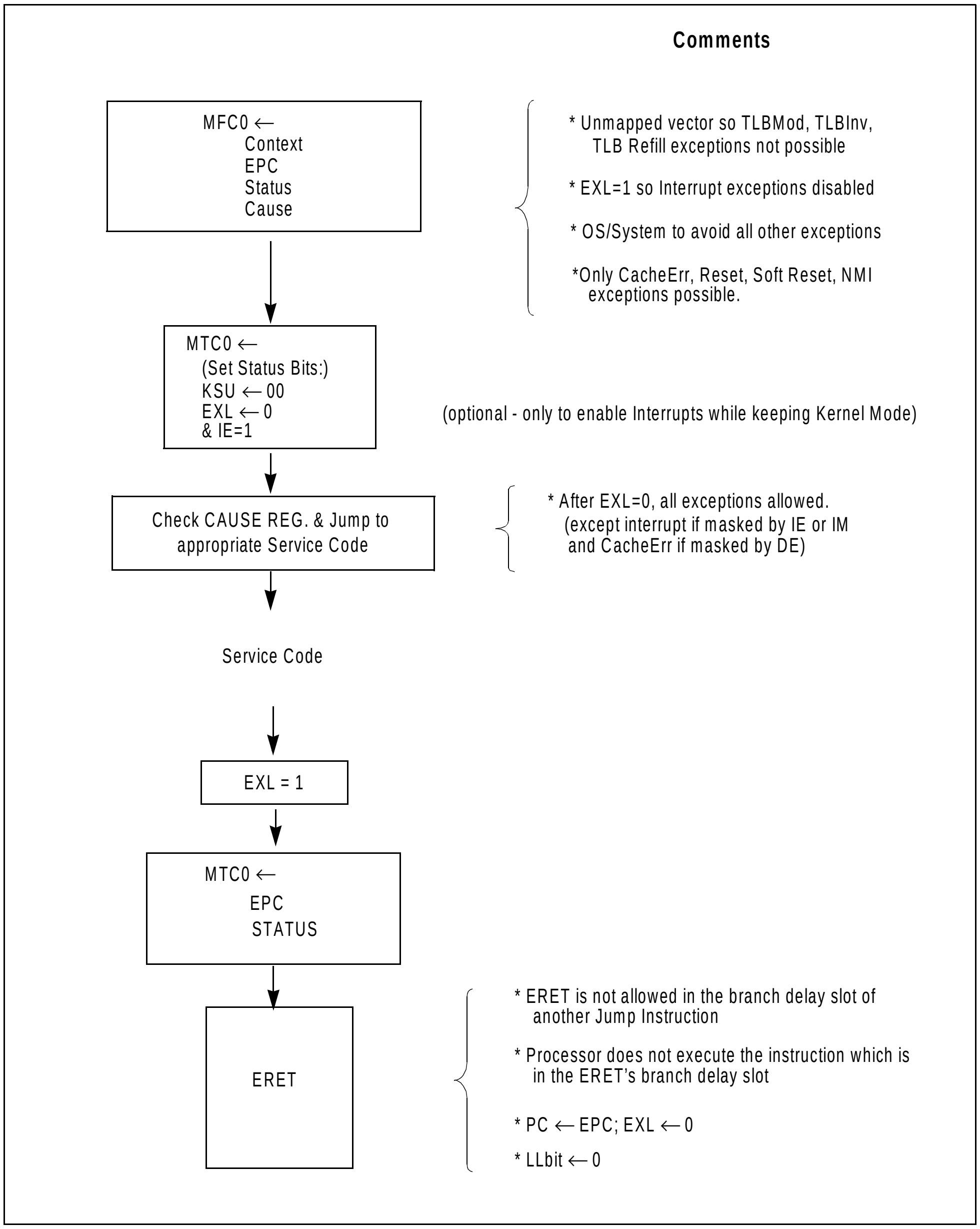

Comments
MFC0 ←
Context
EPC
Status
Cause
* Unmapped vector so TLBMod, TLBInv, TLB Refill exceptions not possible
* EXL=1 so Interrupt exceptions disabled
* OS/System to avoid all other exceptions
*Only CacheErr, Reset, Soft Reset, NMI exceptions possible.
MTC0 ←
(Set Status Bits:)
KSU ← 00
EXL ← 0
& IE=1
(optional - only to enable Interrupts while keeping Kernel Mode)
Check CAUSE REG. & Jump to appropriate Service Code
* After EXL=0, all exceptions allowed. (except interrupt if masked by IE or IM and CacheErr if masked by DE)
Service Code
EXL = 1
MTC0 ←
EPC
STATUS
ERET
* ERET is not allowed in the branch delay slot of another Jump Instruction
* Processor does not execute the instruction which is in the ERET's branch delay slot
* PC ← EPC; EXL ← 0
* LLbit ← 0

Figure 5.19 General Exception Servicing Guideline (SW)

Notes

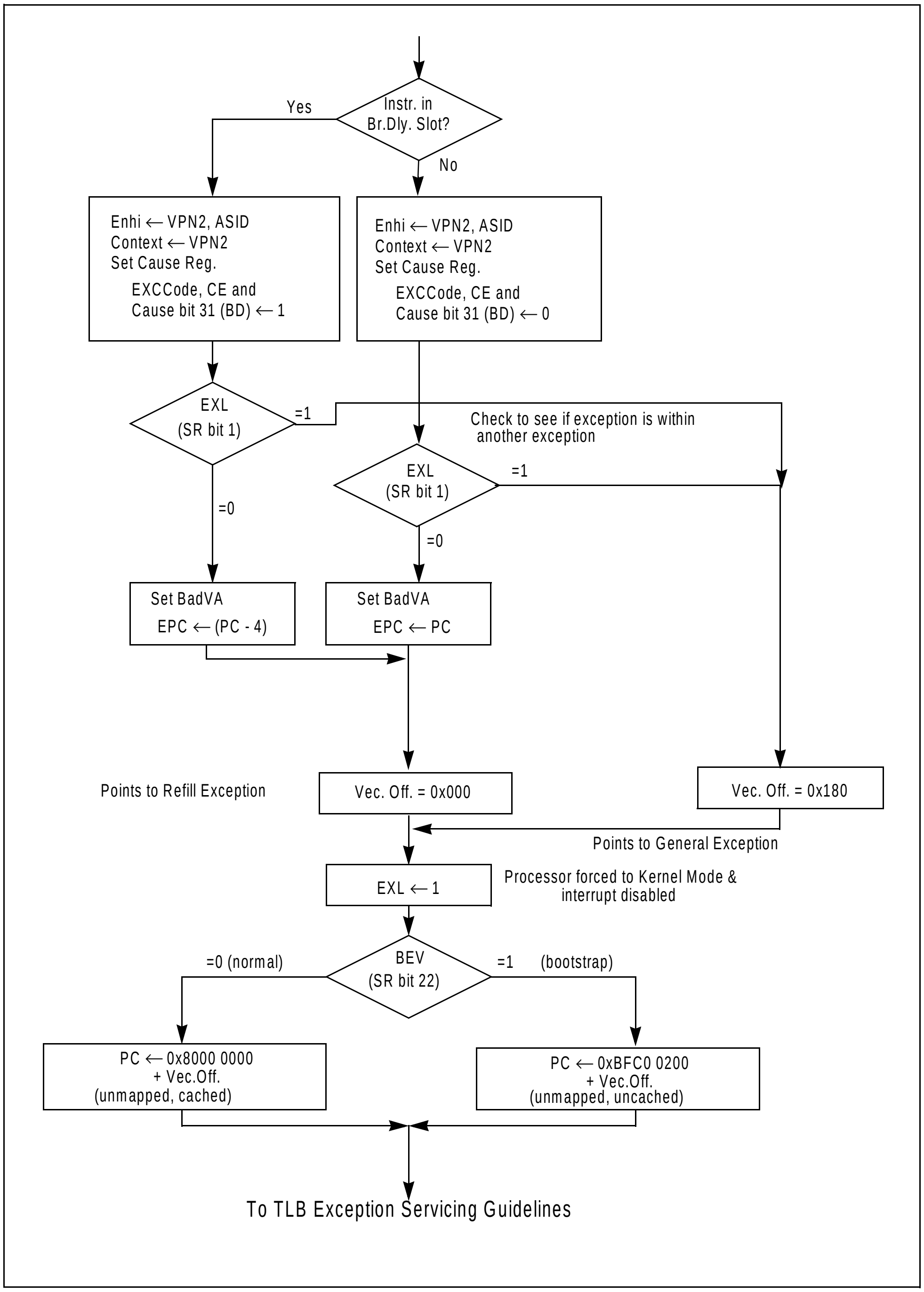
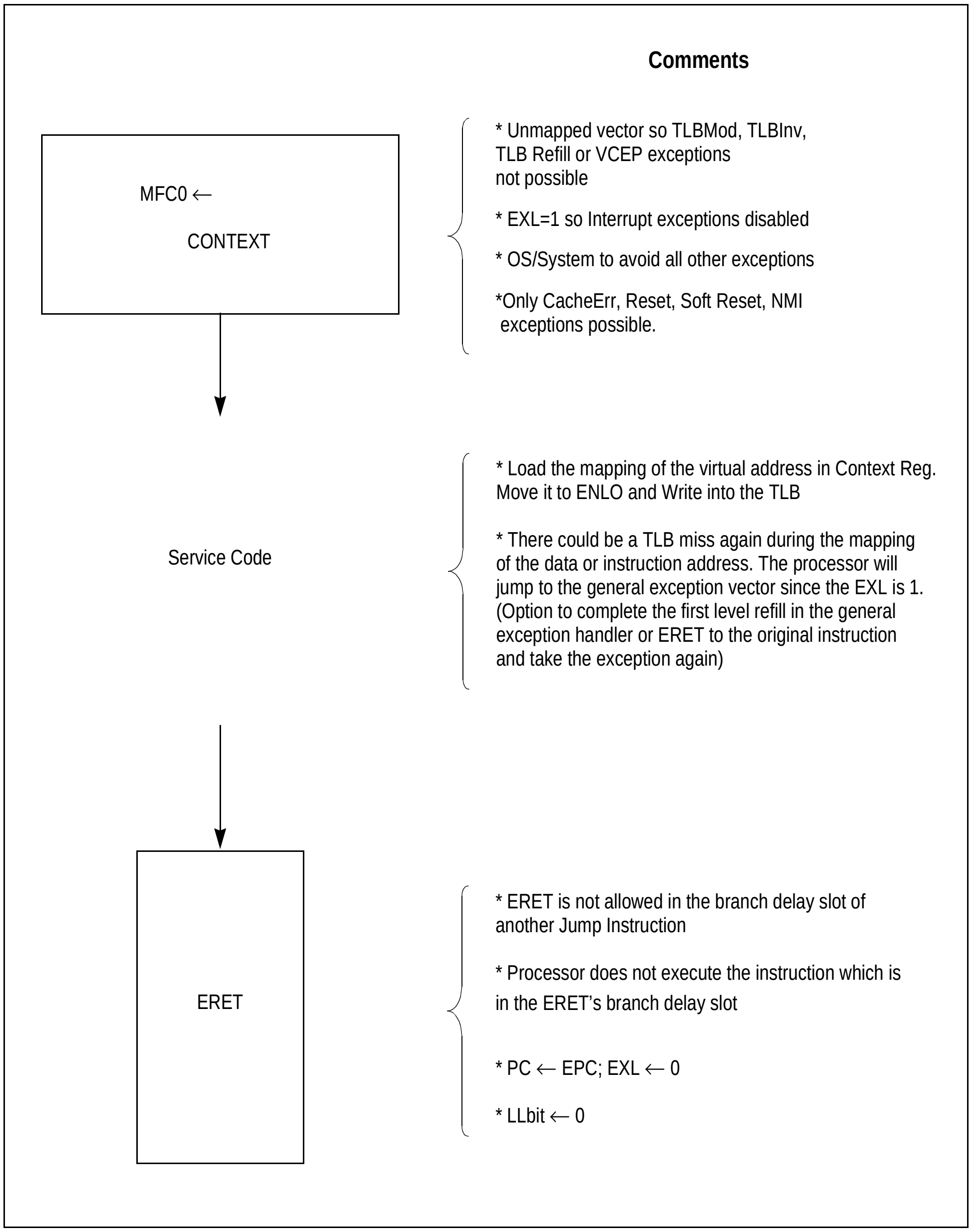


Figure 5.20 TLB Refill Exception Handling (HW)



* Unmapped vector so TLBMod, TLBInv, TLB Refill or VCEP exceptions not possible

* EXL=1 so Interrupt exceptions disabled

* OS/System to avoid all other exceptions

*Only CacheErr, Reset, Soft Reset, NMI exceptions possible.

* Load the mapping of the virtual address in Context Reg. Move it to ENLO and Write into the TLB

* There could be a TLB miss again during the mapping of the data or instruction address. The processor will jump to the general exception vector since the EXL is 1. (Option to complete the first level refill in the general exception handler or ERET to the original instruction and take the exception again)

* ERET is not allowed in the branch delay slot of another Jump Instruction

* Processor does not execute the instruction which is in the ERET's branch delay slot

* PC ← EPC; EXL ← 0

* LLbit ← 0

Figure 5.21 TLB Refill Exception Servicing Guideline (SW)

Notes

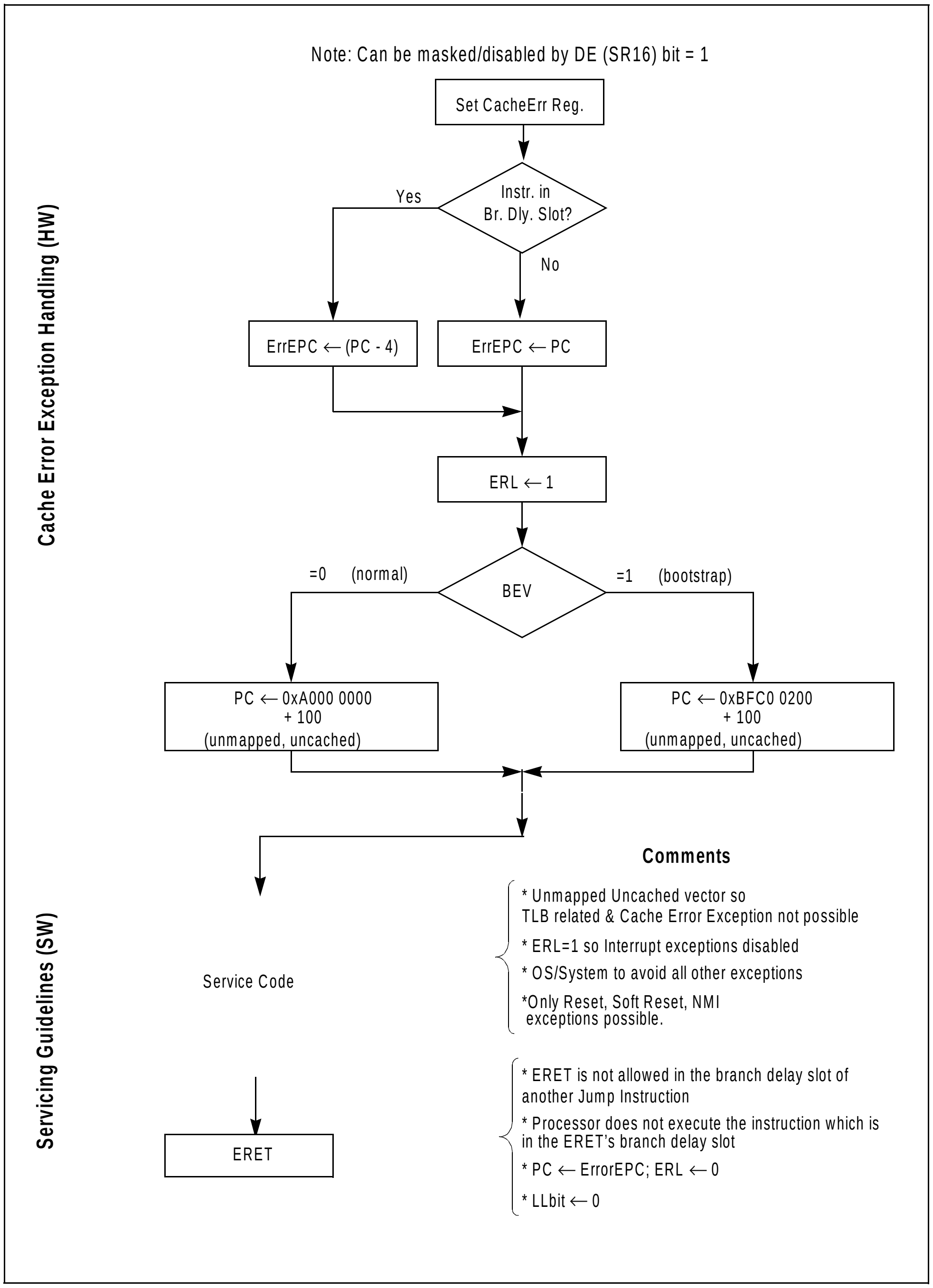

Note: Can be masked/disabled by DE (SR16) bit = 1
Set CacheErr Reg.
Instr. in Br. Dly. Slot?
Yes
No
ErrEPC ← (PC - 4)
ErrEPC ← PC
ERL ← 1
BEV
=0 (normal)
=1 (bootstrap)
PC ← 0xA000 0000 + 100 (unmapped, uncached)
PC ← 0xBFC0 0200 + 100 (unmapped, uncached)
Cache Error Exception Handling (HW)
Servicing Guidelines (SW)
Service Code
ERET
Comments
* Unmapped Uncached vector so TLB related & Cache Error Exception not possible
* ERL=1 so Interrupt exceptions disabled
* OS/System to avoid all other exceptions
*Only Reset, Soft Reset, NMI exceptions possible.
* ERET is not allowed in the branch delay slot of another Jump Instruction
* Processor does not execute the instruction which is in the ERET's branch delay slot
* PC ← ErrorEPC; ERL ← 0
* LLbit ← 0

Figure 5.22 Cache Error Exception Handling (HW) and Servicing Guidelines (SW)

Notes

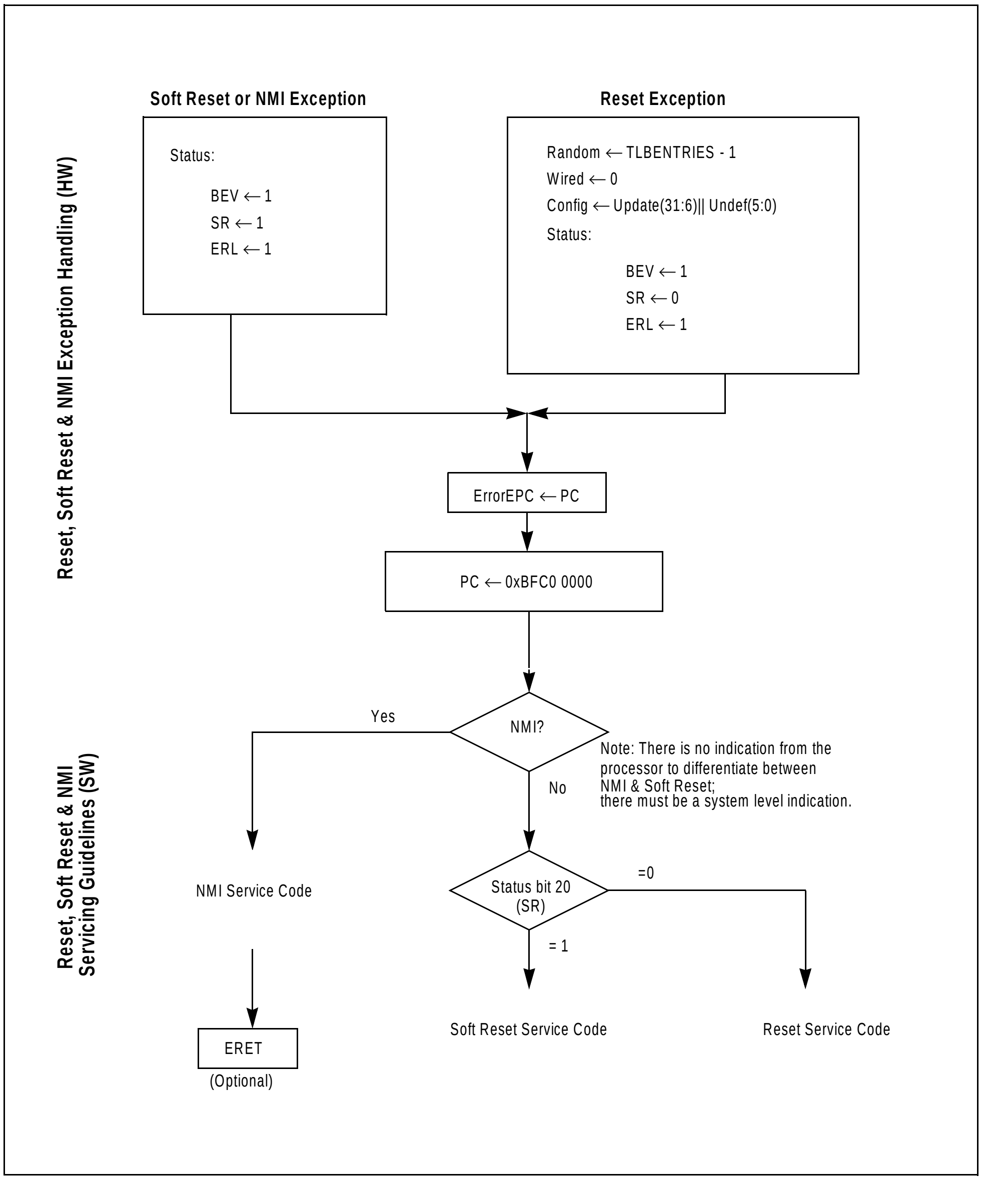

Reset, Soft Reset & NMI Exception Handling (HW)
Reset, Soft Reset & NMI Servicing Guidelines (SW)
Soft Reset or NMI Exception
Status:
BEV ← 1
SR ← 1
ERL ← 1
Reset Exception
Random ← TLBENTRIES - 1
Wired ← 0
Config ← Update(31:6)|| Undef(5:0)
Status:
BEV ← 1
SR ← 0
ERL ← 1
ErrorEPC ← PC
PC ← 0xBFC0 0000
NMI?
Yes
No
Note: There is no indication from the processor to differentiate between NMI & Soft Reset; there must be a system level indication.
NMI Service Code
ERET
(Optional)
Status bit 20 (SR)
=0
= 1
Soft Reset Service Code
Reset Service Code

Figure 5.23 Reset, Soft Reset & NMI Exception Handling (HW) and Servicing Guidelines (SW)

Notes

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 6.1 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.

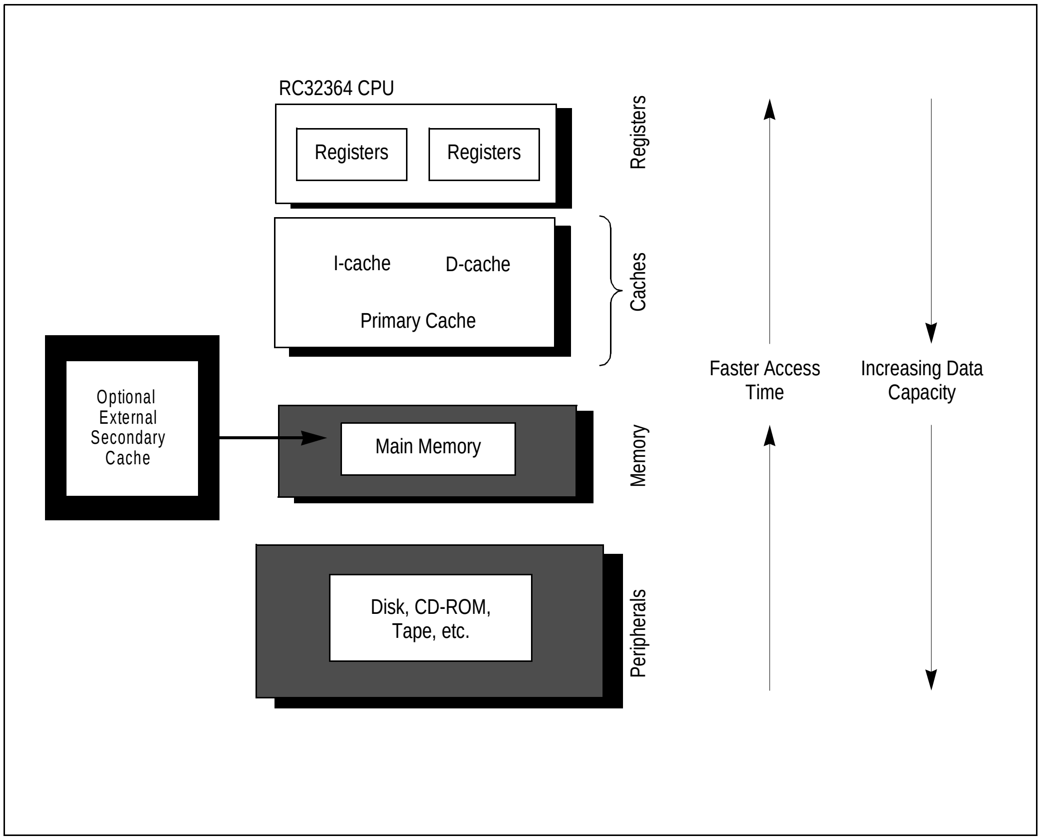


Figure 6.1 Logical Hierarchy of Memory

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 6.4 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.

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).

RC32364 Cache Description

Details of the RC32364'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.

RC32364 Cache Attributes

Table 6.1 highlights the user attributes of the RC32364 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 6.1 RC32364 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.

The primary I-cache of the RC32364 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 6.2 shows the format of a primary I-cache register, and Table 6.2 lists field content descriptions.

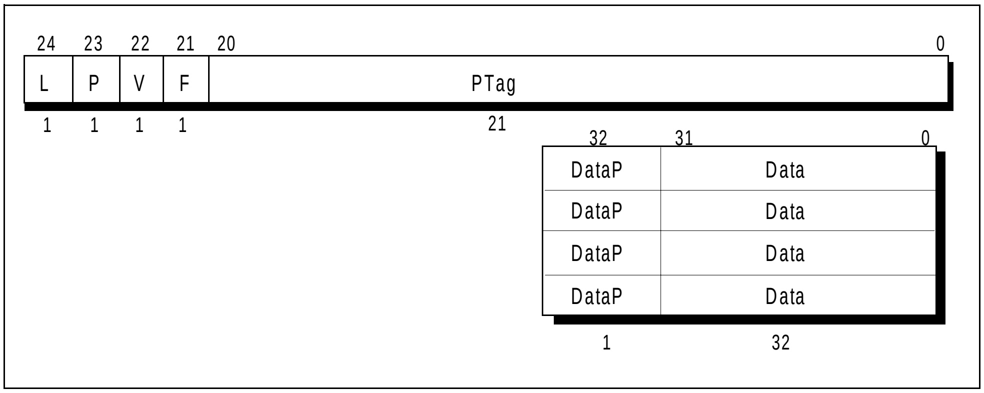


Figure 6.2 Primary I-Cache Line Format

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 6.2 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.

[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

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 RC32364 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*

Figure 6.3 shows the format of a primary D-cache line, Table 6.3 provides the field content descriptions.

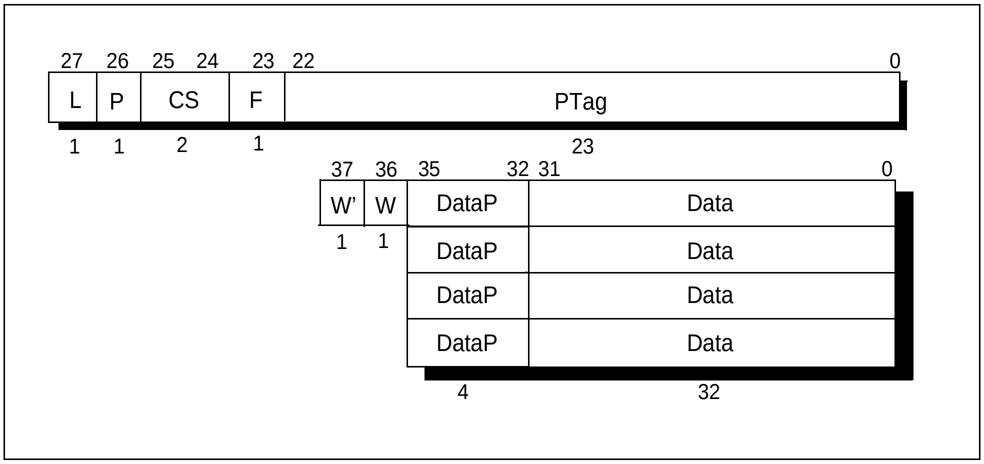


Figure 6.3 Primary D-Cache Line Format

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 6.3 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.

Notes

In the RC32364, 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.

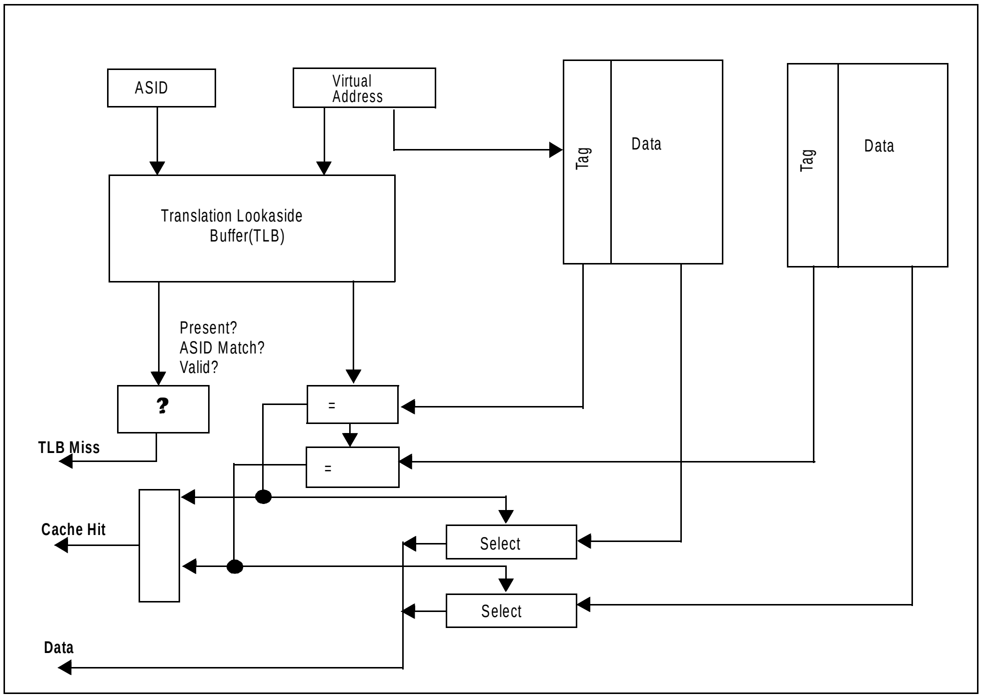


Figure 6.4 Conceptual Primary Cache Lookup Sequence

Accessing the Primary Caches

Figure 6.5 shows the virtual address (VA) index into the primary caches. For the RC32364 an instruction cache is 8kb and the data cache is 2kb.

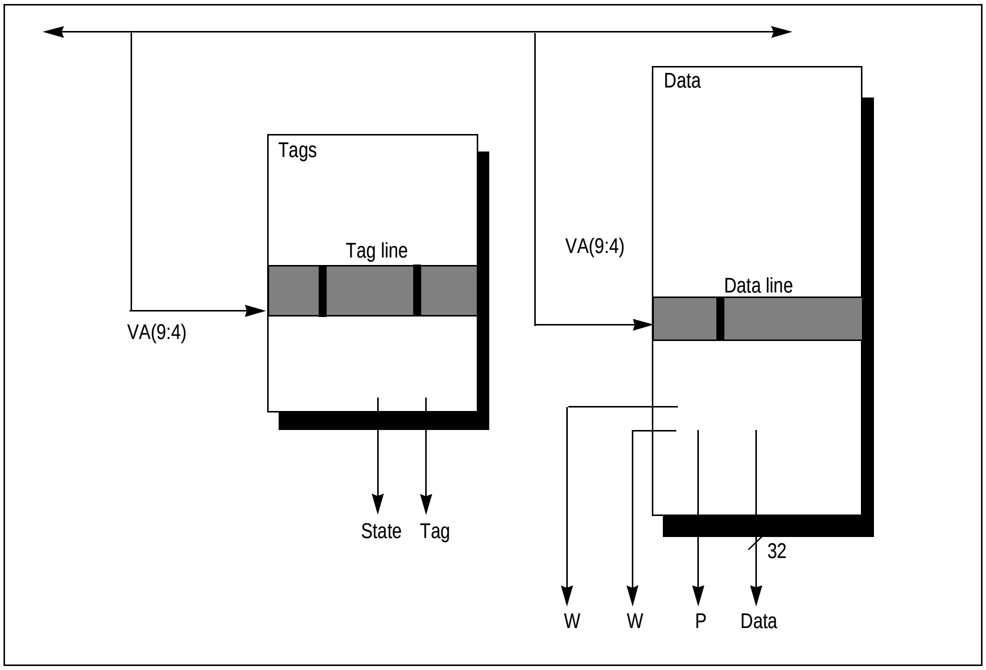


Figure 6.5 Primary Cache Data and Tag Organization

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 RC32364. 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 RC32364 does not provide for hardware cache coherency. This state will not occur during normal operations.*

The RC32364 supports the four cache states shown in Table 6.4. Under normal operations, the only states that will occur in the RC32364, 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 RC32364 system is in one of the states described in Table 6.4.

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 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 in this chapter). Use the W bit to determine if the line must be written back on replacement.

Table 6.4 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*

[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

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.*
- *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 RC32364 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 RC32364 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 RC32364 implements the same store buffer concept that is found in the RC4000. This avoids extra stalls after store instructions to complete the read-modify-write sequence required to update the cache line.

[1] See Table 4.1 in Chapter 4 of this manual for bit values and attribute assignment.

Notes

Cache Replacement Policy

The RC32364 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*[1]*.*
- *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.*

Cache Initialization

The RC32364 includes 2kB of 2-way set associative data cache that corresponds to an address range between 0x000 and 0x7fc. The address offset for set A is 0x000, while the address 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 RC32364 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 RC32364 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 RC32364, 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.

[1] This is an erroneous condition; however, the RC32364 handles this case deterministically.

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:

```
        .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 */
```

Notes

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 */
        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 */
```

Processor Signal Descriptions

Notes

Introduction

This chapter provides easy reference tables on the signals used by and in conjunction with the RC32364 processor. For this device, these signals include the System Interface, the Clock/Control Interface, the Interrupt Interface, Handshake, DMA Interface, and the Initialization Interface.

In this chapter, as well as throughout the manual, active-low signals are denoted with a trailing asterisk; for example, the active-low Cycle-in-progress signal is listed as **CIP***. The tables or figures that follow also identify the signal type as Input (the processor receives it), Output (the processor sends it out) or both (Input and Output).

Figure 7.1 illustrates the functional groupings of the processor signals.

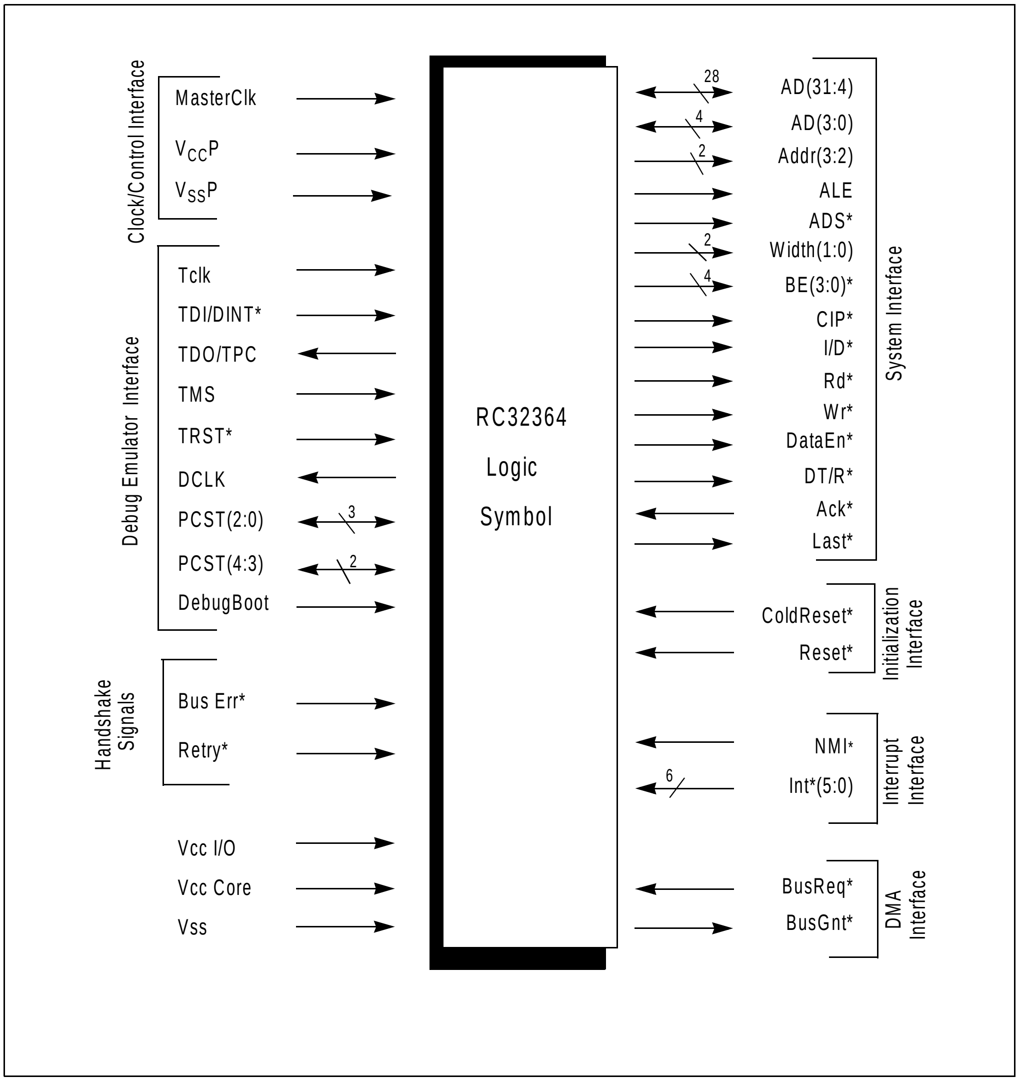


Figure 7.1 IDT79RC32364 Processor Signals

Notes

System Interface Signals

System interface signals provide the connection between the RC32364 processor and the other system components. Table 7.1 lists the system interface signals.

<table>
<tr><th>Pin Name</th><th>Definition</th><th>Direction</th><th>Signal Description</th></tr>
<tr><td>AD(31:4)</td><td>Address/Data</td><td>Input/Output</td><td>Addr(31:4)/Data(31:4)
High-order multiplexed address and data bits. Regardless of system byte ordering, AD(31) is the MSB of the address.</td></tr>
<tr><td>AD(3:0)</td><td>Address/ Data</td><td>Input/Output</td><td>Size(3:0)/Data(3:0)
Transfer size encoding of valid transfer sizes.
For the RC32364, valid sizes are shown below.
<table>
<tr><th>Size(3)</th><th>Size(2)</th><th>Size(1)</th><th>Size(0)</th><th>Transfer Width</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>0</td><td>16 bytes</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td>1</td><td>1 byte</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>0</td><td>2 bytes</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>1</td><td>3 bytes</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>0</td><td>4 bytes</td></tr>
</table>
Other encodings allow future generations to service other transfer sizes. During the data phase, AD[3:0] represents the Data(3:0).</td></tr>
<tr><td>Addr(3:2)</td><td>Address</td><td>Output</td><td>Non-multiplexed address lines. These serve as the word within block address for cache refills (Addr(3:2)). The word within block address bits count in a sub-block ordering. The data retrieval sub-block order algorithm is given in Chapter 9, “Processor Signal Descriptions.”</td></tr>
<tr><td>ALE</td><td>Address Latch Enable</td><td>Output</td><td>This signal provides set-up and hold times around the address phase of the AD bus.</td></tr>
<tr><td>ADS*</td><td>Address Strobe</td><td>Output</td><td>This active-low signal indicates valid address and the start of a new bus transaction. The processor asserts ADS* for the entire address cycle. This is the inverse of the ALE signal.</td></tr>
<tr><td>Width(1:0)</td><td>Bus Width</td><td>Output</td><td>Indicates the Physical Memory/IO data bus size for the correct address. Valid encodings are as follows:
<table>
<tr><th>Width(1)</th><th>Width(0)</th><th>Port Width</th></tr>
<tr><td>0</td><td>0</td><td>8 bits</td></tr>
<tr><td>0</td><td>1</td><td>16 bits</td></tr>
<tr><td>1</td><td>0</td><td>32 bits</td></tr>
<tr><td>1</td><td>1</td><td>Reserved</td></tr>
</table></td></tr>
<tr><td>BE*(3:0)</td><td>ByteEnables/
Addr(1:0)</td><td>Output</td><td>This active-low signal indicates which byte lanes are expected to participate in the data transfer.
<table>
<tr><th></th><th colspan="4">Byte Lanes Enabled In Data Transfer</th></tr>
<tr><td>Port Width</td><td>BE(3)</td><td>BE(2)</td><td>BE(1)</td><td>BE(0)</td></tr>
<tr><td>32-bit</td><td>Used</td><td>Used</td><td>Used</td><td>Used</td></tr>
<tr><td>16-bit</td><td>Byte High Enable</td><td>Not Used</td><td>Address Bit 1 (A1)</td><td>Byte Low Enable</td></tr>
<tr><td>8-bit</td><td>Not Used (Driven High)</td><td>Not Used (Driven High)</td><td>Address Bit 1 (A1)</td><td>Address Bit 0 (A0)</td></tr>
</table>
For information on the endianness, byte addresses and data line usage, refer to the data transfer sequence tables given in Chapter 9.</td></tr>
</table>

Table 7.1 System Interface Pin Descriptions (Part 1 of 2)

Notes

Pin Name	Definition	Direction	Signal Description
CIP*	Cycle-in-progress	Output	This active-low signal denotes that a cycle is in progress. Asserted in the address phase and continue asserted until the ACK* for the last data is sampled.
I/D*	Instruction/Data	Output	Indicates that the current cycle is for an instruction (active- high) or data (active-low) transaction.
Rd*	Read	Output	This active-low signal indicates that the current transaction is a read.
Wr*	Write	Output	This active-low signal indicates that the current cycle transaction is a write.
DataEn*	Data Enable	Output	This active-low signal indicates that the AD bus is in a data cycle. DEN* is asserted after the address cycle (starting of data cycle), and deasserted at the end of the last data cycle.
DT/R*	Data Transmit/ Receive	Output	This active-low signal indicates the current direction of the A/D bus during the current data cycle. “High” indicates it is sourced by the CPU, and “Low” indicates an external source (read response data).
Ack*	Acknowledge Data Phase	Input	On read transactions, this active-low signal indicates to the RC32364 that the memory system has placed valid data on the A/D bus, and that the processor may move the data into the on-chip Read Buffer. On a write transaction, this indicates to the RC32364 that the memory system has accepted the data on the A/D bus.
Last*	Last Data Phase	Output	This active-low output signals the last data transfer of a transaction.

Table 7.1 System Interface Pin Descriptions (Part 2 of 2)

Clock/Control Interface Signals

The Clock/Control interface signals make up the interface for clocking and maintenance. Table 7.2 lists the Clock/Control interface signals.

Name	Definition	Direction	Description
MasterClk	Master clock	Input	MasterClock is the input clock that is the bus clock. The core frequency is derived by multiplying this clock up.
VccP	VccP	Input	Quiet Vcc for PLL.
VssP	VssP	Input	Quiet Vss for PLL.

Table 7.2 Clock/Control Interface Signals

Interrupt Interface Signals

The Interrupt interface signals make up the interface that is used by external agents to interrupt the RC32364 processor. Six external interrupts (Int*(5:0)) and one NMI are available on the RC32364. Table 7.3 lists the Interrupt Interface signals.

Name	Definition	Direction	Description
NMI*	Non-Maskable Interrupt	Input	The Non-Maskable Interrupt signal is an asynchronous falling-edge-sensitive signal.
Int*(5:0)	Interrupt/ ModeBit(9:6)	Input	These interrupt inputs are active low to the CPU. During power-on, Int*(3:0) serves as ModeBit(9:6).

Table 7.3 Interrupt Interface Signals

Notes

Initialization Interface Signals

The Initialization Interface signals make up the interface by which an external agent initializes the processor operating parameters. Table 7.4 lists the Initialization Interface signals.

Name	Definition	Direction	Description
Cold-Reset*	Cold reset	Input	This signal must be asserted for a power on reset or a cold reset. **ColdReset*** must be deasserted synchronously with **MasterClock**.
Reset*	Reset	Input	This signal must be asserted for any reset sequence. It can be asserted synchronously or asynchronously for a cold reset, or synchronously to initiate a warm reset. **Reset*** must be deasserted synchronously with **MasterClock**.

Table 7.4 Initialization Interface Signals

DMA Interface Signals

To share the system interface bus, the RC32364 provides the BusReq* and BusGnt* signals to interface external DMA masters. A DMA arbiter provides the external master control of the external bus.

Name	Definition	Direction	Description
BusReq*	Bus Request	Input	This active-low signal is an input to the processor and is used to request mastership of the external interface bus. Mastership is granted according to the assertion of this input and taken back based on its negation.
BusGnt*	Bus Grant	Input/Output	This active-low signal is an output from the processor and is used to indicate that the CPU has relinquished mastership of the external interface bus. BusGnt* goes low initially for at least 2 clocks to indicate that the CPU has relinquished mastership of the external interface bus. After going low, BusGnt* returns high, either when the CPU makes an internal request for the bus or after BusReq* is de-asserted. During the power-on reset (Cold Reset), BusGnt* is an input, ModeBit(5).

Table 7.5 DMA Interface Signals

Handshake Interface

The process of asking a question of a device and receiving information in return is called handshaking. Every communication protocol between one device and the next will implement some form of handshaking. In the RC32364, this type of bus activity is indicated through several signals, including Ack*, Retry* and BusErr*. Also, as noted in the system interface signals, this device provides I/D* signals, to indicate whether an instruction or data is being transferred.

Name	Definition	Direction	Description
BusErr*	Bus Error	Input	Indicates that a bus error has occurred.
Retry*	Retry	Input	Indicates that the current bus cycle must be terminated. Retry* is ignored after acceptance of the first data during a read cycle. During a write, Retry* is recognized in all data cycles.

Table 7.6 Handshake Interface Signals

Debug Emulator Interface

The RC32364 features on-chip support for low-cost in-circuit emulation equipment and implements the standard MIPS enhanced JTAG interface. This interface uses **MasterClock** and other signals as described in Table 7.7.

<table>
<tr><th>Name</th><th>Definition</th><th>Direction</th><th>Description</th></tr>
<tr><td>Tclk</td><td>Test Clock</td><td>Input</td><td>An input test clock, used to shift into or out of the boundary-Scan register cells. Tclk is independent of the system and the processor clock with nominal 50% duty cycle.</td></tr>
<tr><td>TDI/DINT*</td><td>Test Data Input/ DINT*</td><td>Input</td><td>On the rising edge of Tclk, serial input data is 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)</td></tr>
<tr><td>TDO/TPC</td><td>Test Data Output/ TPC</td><td>Output</td><td>The TDO signal provides serial data shifted out from the instruction or data register on the falling edge of Tclk. When no data is shifted out, the 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.</td></tr>
<tr><td>TMS</td><td>Test Mode Select</td><td>Input</td><td>The logic signal received at the TMS input is decoded by the TAP controller to control test operation. TMS is sampled on the rising edge of the TCLK.</td></tr>
<tr><td>TRST*</td><td>Test Reset Input</td><td>Input</td><td>The TRST* pin is an active-low signal for asynchronous reset of the debug unit, independent of the processor logic. Requires an external pull-down on the board.</td></tr>
<tr><td>DCLK</td><td>Processor Clock</td><td>Output</td><td>During Real Time Mode, this signal is used to capture address and data from the TDO signal at the processor clock speed or any division of the internal pipeline. DCLK will be at 1/3 of the pipeline clock.</td></tr>
<tr><td>PCST(2:0)/ ModeBit(2:0)</td><td>PC Trace Status Information</td><td>Input/Output</td><td>These bits indicate processor clock activity as follows:
<table>
<tr><th>Bit Value</th><th>Operation</th></tr>
<tr><td>111</td><td>Pipeline Stall (STL)</td></tr>
<tr><td>110</td><td>Branch/Jump forms with PC output (JMP)</td></tr>
<tr><td>101</td><td>Branch/Jump forms with no PC output (BRT)</td></tr>
<tr><td>100</td><td>Exception generated with an exception vector code output (EXP)</td></tr>
<tr><td>011</td><td>Sequential performance (SEQ)</td></tr>
<tr><td>010</td><td>Trace is outputted at pipeline stall time (TST)</td></tr>
<tr><td>001</td><td>Trace trigger output at performance time (TSQ)</td></tr>
<tr><td>000</td><td>Run Debug Mode (DBM)</td></tr>
</table>
During power-on reset (cold reset), PCST(2:0) serves as ModeBit(2:0).</td></tr>
</table>

Table 7.7 ICE/Debug Interface Signals (Part 1 of 2)

Notes

Name	Definition	Direction	Description
PCST(4:3/ ModeBit(4:3)	PC Trace Status Information	Input/Output	Reserved. During power-on reset, PCST(4:3) serves as ModeBit(4:3).
DebugBoot	Debug Boot Unit	Input	The Debug Boot signal 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 an 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 7.7 ICE/Debug Interface Signals (Part 2 of 2)

Clocking, Reset and Initialization Interfaces

Notes

Introduction

This chapter provides a description of the clock signals ("clocks") that are used on the RC32364 processor and includes a discussion on the basic system clocks and system timing parameters. A brief explanation on the power reduction modes for this device is also presented.

The reset and initialization interface is also discussed and includes tables listing boot-mode configuration settings with timing diagrams for the warm and cold sequences.

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 8.1 and Figure 8.2.

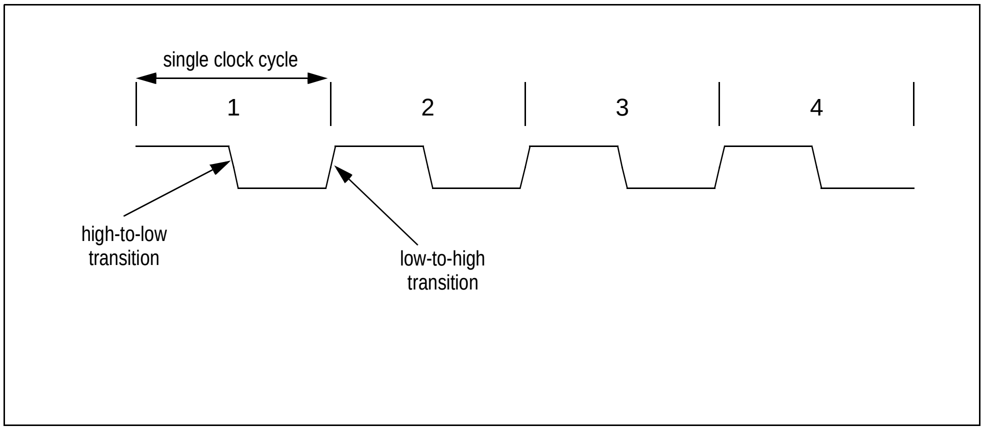


Figure 8.1 Signal Transitions

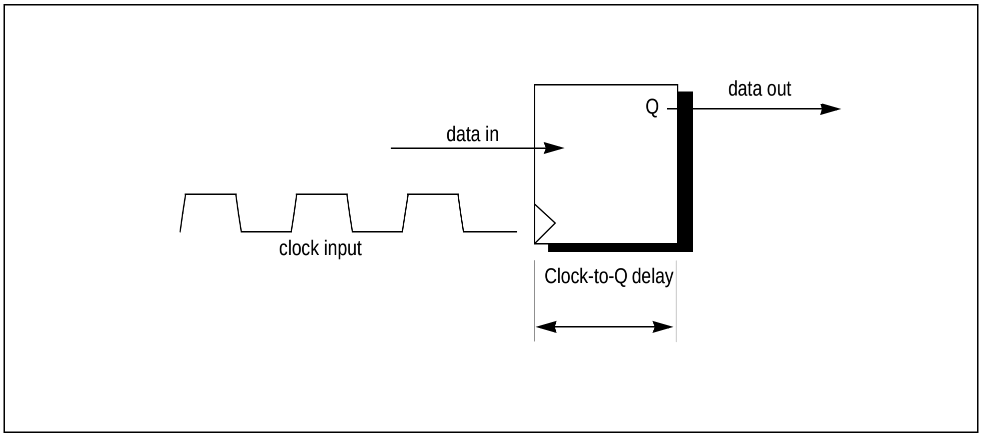


Figure 8.2 Clock-to-Q Delay

Notes

Basic System Clocks

The RC32364 processor has a single input clock, **MasterClock**, and no output clocks.

MasterClock

The **MasterClock** input must meet the maximum rise time (T_{MCRise}), maximum fall time (T_{MCfall}), minimum clock high (T_{MHIGH}) time, minimum clock low (T_{MCLOW}) time, and input jitter ($T_{JitterIn}$) parameters for proper phase locked loop (PLL) operation.

The processor bases all internal clocking on the single **MasterClock (MClk)** input signal. The RC32364 uses **MasterClock** to sample data at the system interface and to clock data into the processor system interface output register.

The external agent should use **MasterClock** for the global system clock and for clocking the output registers of an external agent. Figure 8.3 shows the input, output and hold time parameters measured at the midpoint of the rising clock edge.

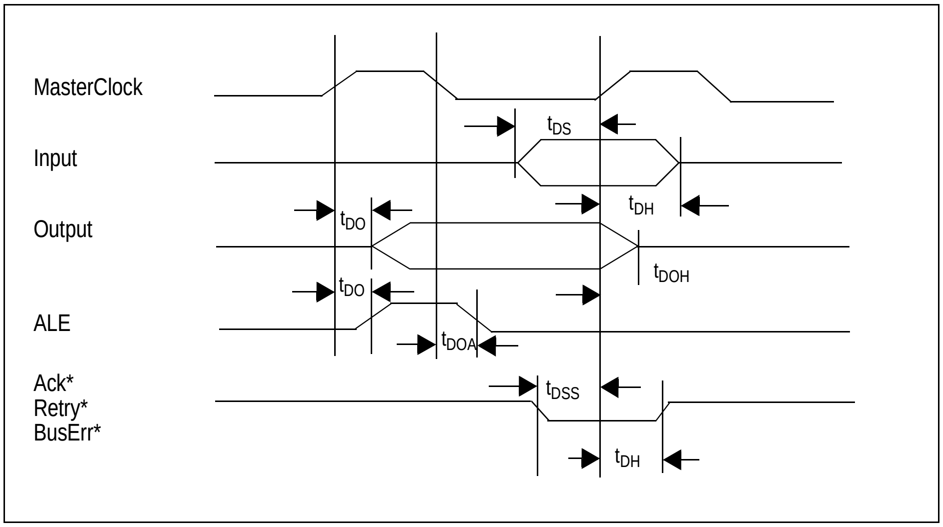


Figure 8.3 System Clocks Data Setup, Output, and Hold timing

PClock

By multiplying **MasterClock** 2, 3, 4, 5, 6, 7, or 8 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 8.4 shows the clocks for a MasterClock-to-PClock multiply by 2.

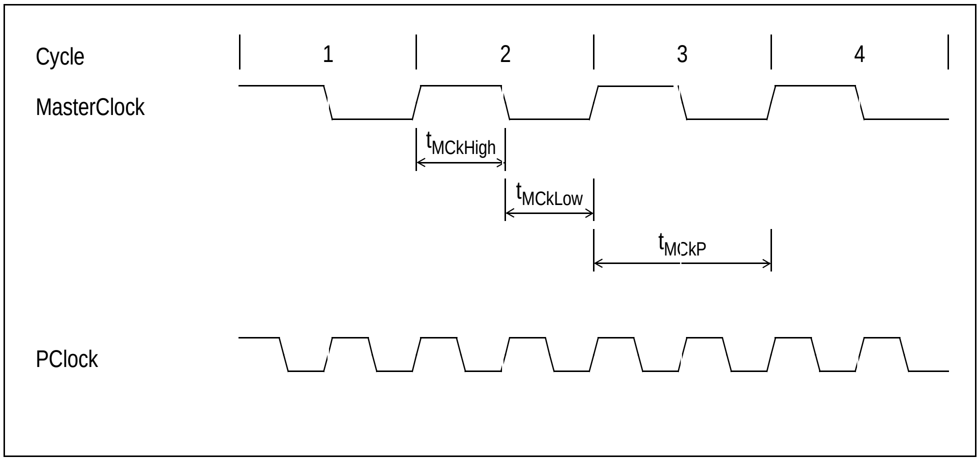


Figure 8.4 Timing Illustration of MasterClock-to-PClock Multiply by 2

Notes

Phase-Locked Loop (PLL) Operation

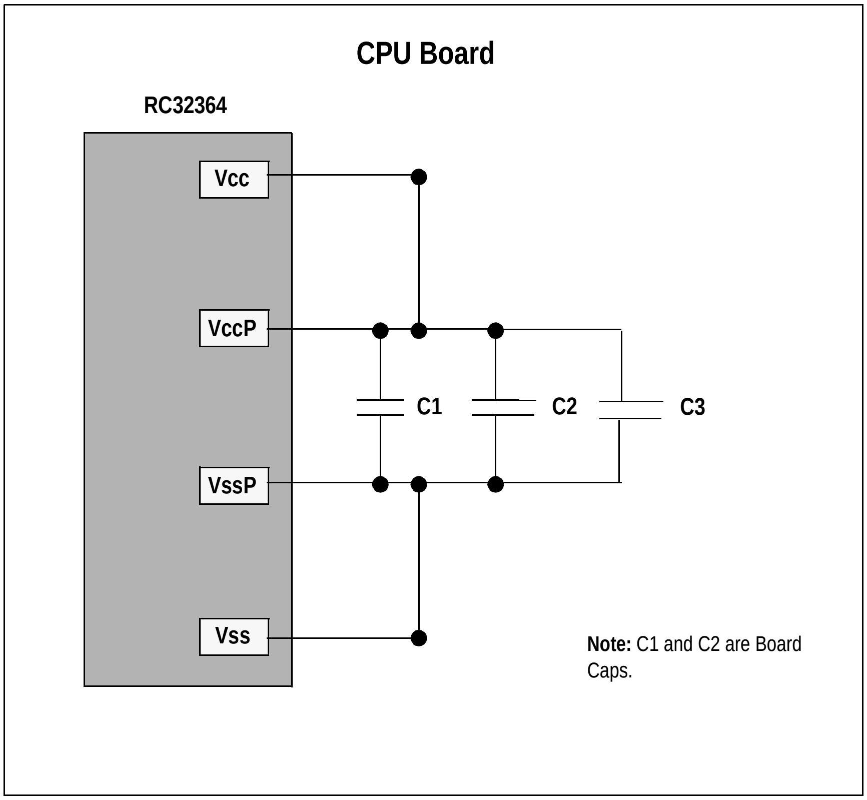
The processor aligns the pipeline clock, **PClock**, to the **MasterClock** by using an internal phase-locked loop (PLL) circuit that generates aligned clocks. By their nature, PLL circuits are only capable of generating aligned clocks for **MasterClock** frequencies within a limited range.

Clocks generated using PLL circuits contain some inherent inaccuracy, or jitter; a clock aligned with **MasterClock** by the PLL can lead or trail **MasterClock** 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 RC32364. 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 8.5.



Note: C1 and C2 are Board Caps.

Figure 8.5 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.

Notes

PLL Analog Power Filtering

For noisy module environments, a filter circuit of the following form is recommended as shown in Figure 8.6.

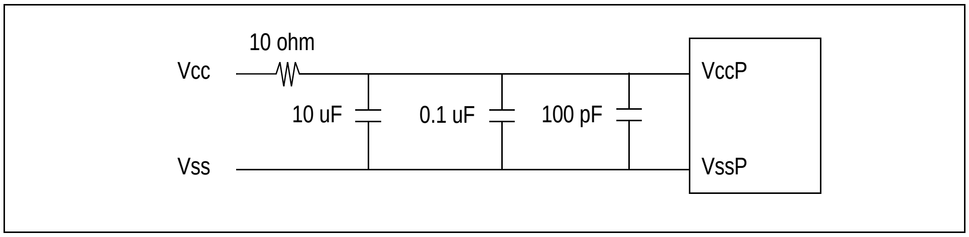


Figure 8.6 PLL Filter Circuit for Noisy Environments

Reset and Initialization Interface

During the reset sequence, the RC32364 obtains configuration information using its mode configuration interface. The initialization values for the RC32364 are obtained from PCST[4:0], BusGnt* and Int*[4:0], which are ModeBit[9:0] during the power-on reset. The ModeBit[9:0] are latched with the rising edge (negating edge) of the ColdReset* signal. Timing of the mode configuration interface reset sequence is shown in Figure 8.7 on page 8-4.

Note: All signals, except Bus Request and Bus Grant, are tristated during cold reset.

The boot-mode configuration settings are listed in Table 8.1.

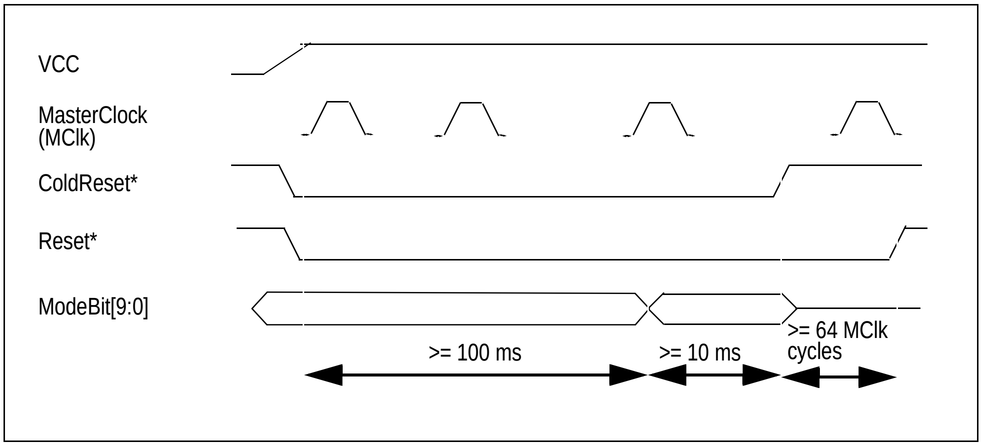


Figure 8.7 Mode Configuration Interface Reset Sequence

Notes

Boot-Mode Configuration Settings

Pin	Mode Bit	Description	Value	Mode Setting
PCST[2:0]	2:0 MSB (2)	Clock Multiplier **MasterClock** is multiplied internally to generate PClock	0	Multiply by 2
			1	Multiply by 3
			2	Multiply by 4
			3	Multiply by 5
			4	Multiply by 6
			5	Multiply by 7
			6	Multiply by 8
			7	Reserved
PCST[3]	3	EndBit	0	Little-endian ordering
			1	Big-endian ordering
PCST[4]	4	Reserved	0	
BusGnt*	5	Reserved	0	
Int*[0]	6	TmrIntEn Enables/Disables the timer interrupt on Int*[5]	0	Enables timer interrupt
			1	Disables timer interrupt
Int*[1]	7	Reserved for future use.	1	
Int*[3:2]	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 8.1 Boot-Mode Configuration Settings

Notes

Reset Interface

The RC32364 processor has the following two types of resets, which use the ColdReset* and Reset* input signals:

- ***Power-on reset*** *starts when the power supply is turned on and completely reinitializes 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 8.8.*
- ***Warm reset*** *restarts the processor but does not affect the clocks. The processor preserves the internal state and takes the reset exception. Timing of the warm reset operation is illustrated in Figure 8.9.*

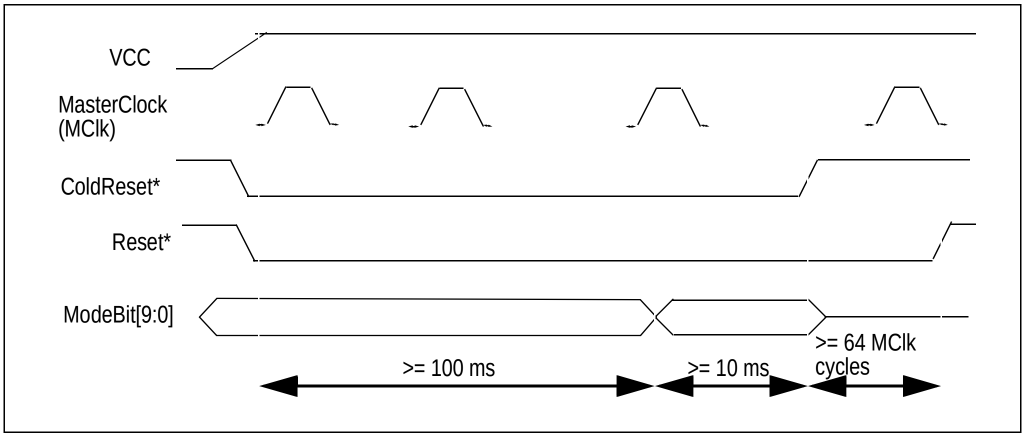


Figure 8.8 Timing of Cold Reset Signal

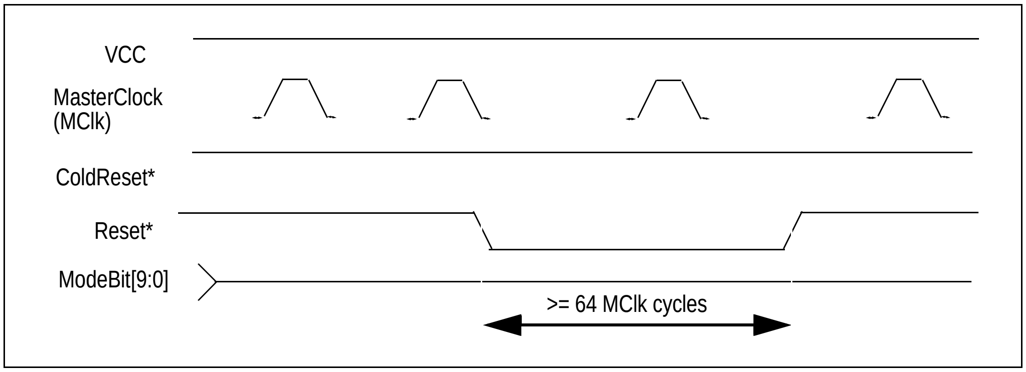


Figure 8.9 Timing of Warm-Reset Signal

Bus Interface Overview

Notes

Introduction

The RC32364's 32-bit data interface is through a multiplexed address and data bus. As with the RC32300 family, the RC32364 implements special techniques for byte-enable generation as well as for low-order address bits in block refills.

Parity is not supported for the bus; however, the internal cache is parity protected. Parity is generated into the cache on refill and then checked when the data is brought to the execution core.

Feature Overview

The following features are enabled by the RC32364 bus interface unit:

- *Support of multiple pipeline-to-system clock ratios*
- *Support of variable port-width interface, including 8-bit boot prom*
- *Multiplexed address and data bus*
- *Sub-block ordering read and sequential ordering write*
- *DMA arbiter interface*
- *Input clock is bus clock: core frequency derived from there*

Data Interfaces

Table 9.1 lists the signal names and descriptions for the data interface pins of the RC32364.

Pin	Signal	Description
AD(31:0)	Data(31:0)	32 data bits multiplexed on to the same bus as the address bus.

Table 9.1 Data Interface Pin Description

Variable Port-Width Interface

The RC32364 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 RC32364 reset-mode initialization interface supports the setting of the boot-prom port width.

To simplify design, the RC32364 elects to use the same data lines, for a given width of memory, regardless of memory byte-ordering (endianness[1]). Table 9.2 lists which byte lanes are used and Table 9.3, Table 9.4, and Table 9.5 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 9.2 Port Width Assignments to Data Lines

[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
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 9.3 Data Transfer Sequences for 8-bit Port Width

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 9.4 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)

Table 9.5 Data Transfer Sequences for 32-bit Port Width (Part 1 of 2)

Notes

Port Width	Transfer Size	Byte Address	Endianness	Data Lines
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 9.5 Data Transfer Sequences for 32-bit Port Width (Part 2 of 2)

Interface Control Registers

The following three interface control registers are used in the RC32364:

- *The Port-Width Control register controls attributes of the various memory systems and is used to interface the RC32364 to varying width memory regions.*
- *The Bus Turnaround (BTA) control register controls the bus turnaround cycle(s) for the various memory systems. The RC32364 divides the physical address space into various sub-regions, each of which features independently programmable bus turnaround cycle(s).*
- *The Bus Error Address Register holds the physical address of the transfer that signalled the most recent bus error.*

The Port-Width Control Register: Virtual Address 0xFFFF_E200

The RC32364 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 RC32364 to manage the bus interaction with the actual memory system to satisfy this model.

The format of the Port Width Control register is shown in Figure 9.1. Table 9.6 lists the register's fields and content descriptions.

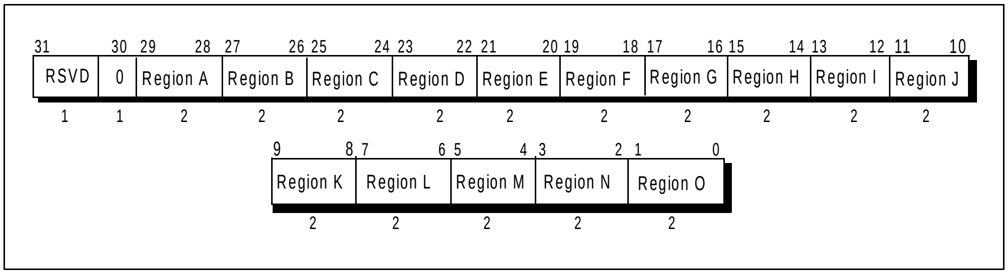


Figure 9.1 Format of Port Width Control Register

Notes

Field	Description
0	Reserved
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
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 9.6 Port Width Control Register Field Definition

Width fields are encoded as shown in Table 9.7.

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

Table 9.7 Encoding of 8-, 16-, and 32-bit Port Widths

The address ranges served by each named region are listed in Table 9.8. 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.*

Notes

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
RegionL	100x xx	512MB
RegionM	101x xx	512MB
RegionN	110x xx	512MB
RegionO	111x xx	512MB

Table 9.8 Memory Region Address Ranges

The 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 9.2. This register’s fields and content descriptions are listed in Table 9.9.

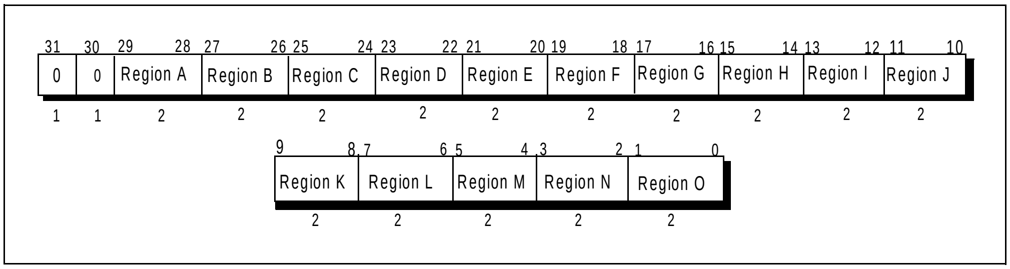


Figure 9.2 Bus Turnaround (BTA) Control Register Format

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

Table 9.9 Bus Turnaround (BTA) Control Register Field Descriptions (Part 1 of 2)

Notes

Field	Definition
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
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 9.9 Bus Turnaround (BTA) Control Register Field Descriptions (Part 2 of 2)

The turnaround cycle(s) is encoded as shown in Table 9.10. Figure 9.3 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

Table 9.10 Width Encoding of Bus Turnaround Cycles

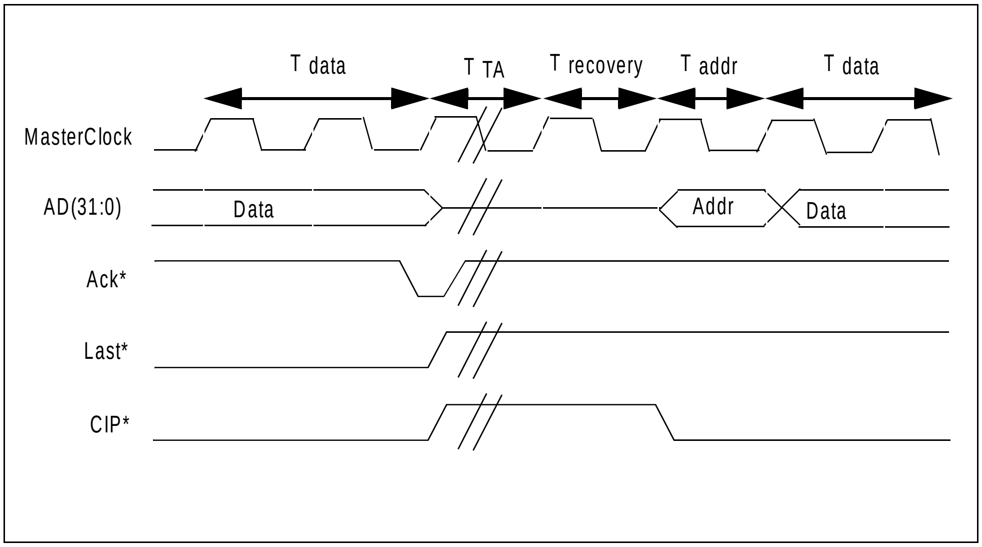


Figure 9.3 Timing of Bus Turnaround Cycle(s)

The 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.

The Address Interface

The address information of a transfer is primarily presented on the same multiplexed address/data bus as the data lines. However, additional information is provided to simplify system design.

This information relates to low order address bits (byte within word for narrow interfaces; word within block for cache refill); transfer size information; byte-enable signals; address demux control; etc.

Address Signal Description

Table 9.11 lists the signals provided for addressing operations. Pin names followed by an asterisk (*) are active when low.

<table>
<tr><th>Pin</th><th>Signal</th><th>Description</th></tr>
<tr><td>AD(31:4)</td><td>Addr(31:4)/
Data(31:4)
(I/O)</td><td>High-order multiplexed address and data bits.
Regardless of system byte ordering, AD(31) is the MSB of the address.</td></tr>
<tr><td>AD(3:0)</td><td>Size(3:0)
(I/O)</td><td>Transfer size encoding of valid transfer sizes. For the RC32364, valid sizes are shown in the table below.</td></tr>
<tr><td>Addr(3:2)</td><td>Addr(3:2)
(O)</td><td>Non-multiplexed address lines. These also serve as the word within block address for cache refills (Addr(3:2)). The word within block address bits count in sub-block ordering for read and sequential for write.</td></tr>
<tr><td>ALE</td><td>ALE
(O)</td><td>Address Latch Enable. This signal provides set-up and hold time around the address phase of the AD bus.</td></tr>
<tr><td>Width(1:0)</td><td>Bus Width
(O)</td><td>Indicates the Physical Memory/IO data bus size.
Valid sizes are shown in Table 9.13 on page 9-8.</td></tr>
<tr><td>CIP*</td><td>Cycle-in-progress
(O)</td><td>Denotes that a cycle is in progress. Asserted in the address phase and continues asserted until the ACK* for the last data is sampled.</td></tr>
<tr><td>I/D*</td><td>I/D*
(O)</td><td>Indicates that the current cycle is for an instruction (active high) or data (active low) transaction.</td></tr>
<tr><td>Rd*</td><td>Read
(O)</td><td>This active-low signal indicates that the current transaction is a read.</td></tr>
<tr><td>Wr*</td><td>Write
(O)</td><td>This active-low signal indicates that the current cycle transaction is a write.</td></tr>
<tr><td>DT/R*</td><td>Data Transmit/
Receive
(O)</td><td>This active-low signal indicated the current direction of the A/D bus during the current data cycle. “High” indicates it is sourced by the CPU, and “Low” indicates an external source (read response data).</td></tr>
<tr><td>DataEn*</td><td>Data Enable
(O)</td><td>This active-low signal indicates that the AD bus is in a data cycle.
DEN* is asserted after the address cycle (start of 1st data cycle), and deasserted at the end of the last data cycle.</td></tr>
<tr><td>BE*(3:0)</td><td>Byte Enables/
Addr(1:0)
(O)</td><td>Byte enable (3:0)
This active-low signal indicates which byte lanes are expected to participate in the data transfer.
<table>
<tr><th></th><th colspan="4">Byte Lanes Enabled In Data Transfer</th></tr>
<tr><th>Port Width</th><th>BE*(3)</th><th>BE*(2)</th><th>BE*(1)</th><th>BE*(0)</th></tr>
<tr><td>32-bit</td><td>Used</td><td>Used</td><td>Used</td><td>Used</td></tr>
<tr><td>16-bit</td><td>Byte High Enable</td><td>Not Used (Driven High)</td><td>Address Bit 1 (A1)</td><td>Byte Low Enable</td></tr>
<tr><td>8-bit</td><td>Not Used (Driven High)</td><td>Not Used (Driven High)</td><td>Address Bit 1 (A1)</td><td>Address Bit 0 (A0)</td></tr>
</table></td></tr>
</table>

Table 9.11 Addressing Interface Signals

Notes

Valid Transfer Sizes

Table 9.12 lists the encoding for the “Size” (AD(3:0)) field:

Size(3)	Size(2)	Size(1)	Size(0)	Transfer Width
0	0	0	0	16 bytes
0	0	0	1	1 byte
0	0	1	0	2 bytes
0	0	1	1	3 bytes
0	1	0	0	4 bytes
All other encodings are reserved for future use.				

Table 9.12 Encoding for Valid Transfer Sizes

Memory Port Width Encodings

Table 9.13 lists the encoding for 8-,16- and 32- port widths:

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

Table 9.13 Encoding of 8-, 16-, and 32-bit Port Widths

Address Generation Timing

Figure 9.4 shows the address generation phase at the beginning of a transfer.

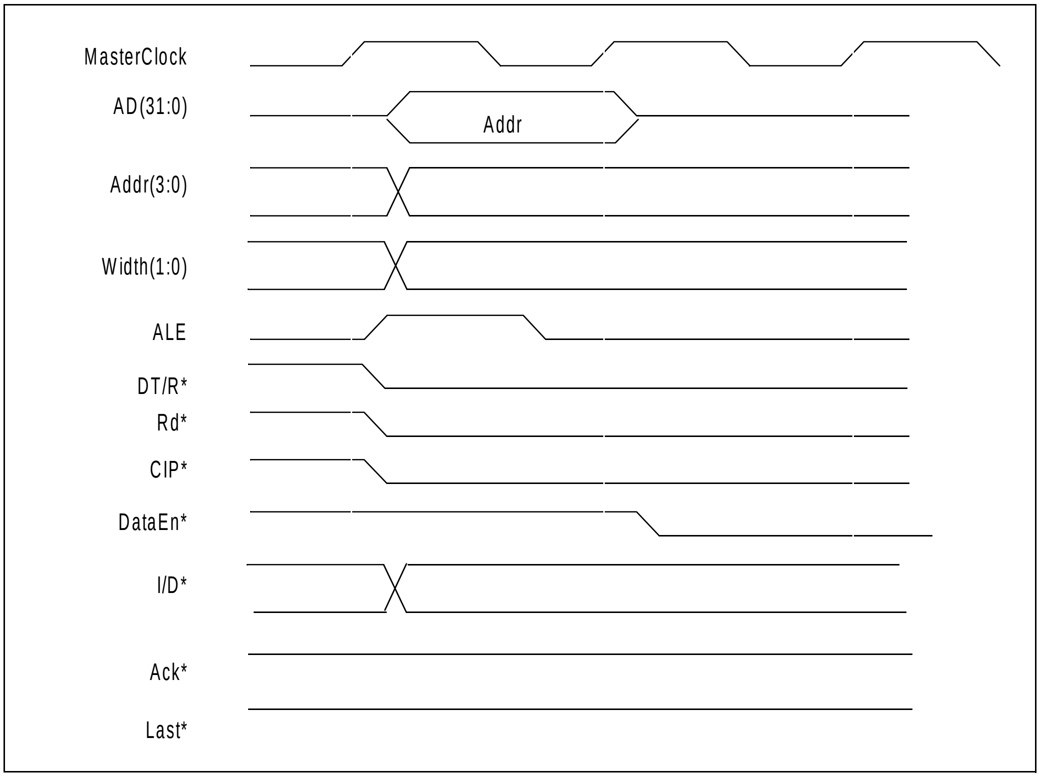


Figure 9.4 Address Generation at Start of Transfer

Notes

Figure 9.5 shows the toggling of Addr(3:0) in the middle of a multiple data transfer.

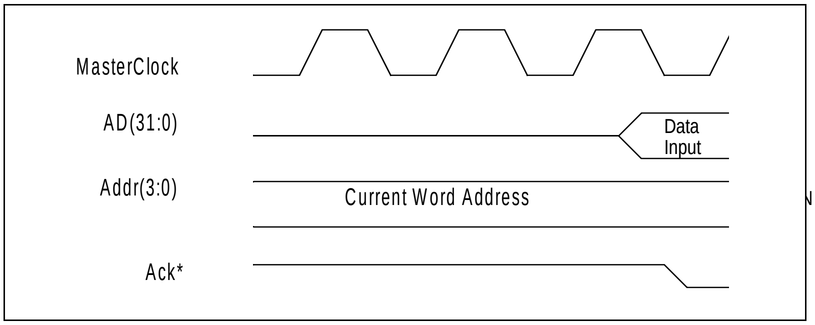


Figure 9.5 Address Toggling Mid-Transfer

Read Control Interface

The read control interface is designed to easily support the following read transactions:

- *a single data read that can be satisfied in one data phase*
- *a single data read that requires multiple data phases (the data is wider than the port)*
- *a cache block read to any of the valid port widths*
- *subblock ordering, accessing among the words of a cache refill*
  - *allows the system to define the order in which the data elements are retrieved (see Figure 9.6[1])*

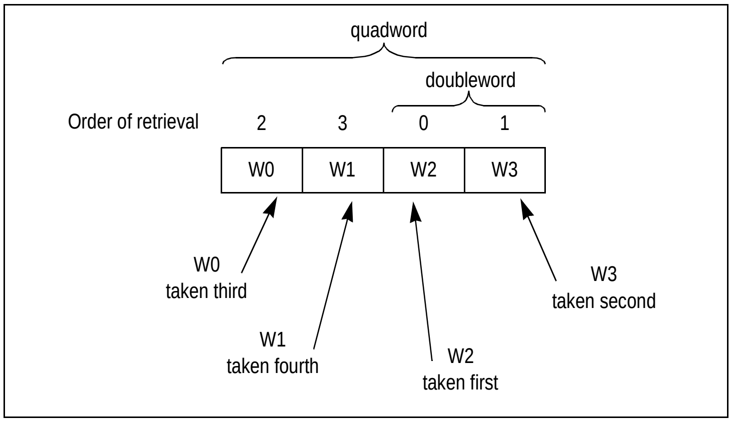


Figure 9.6 Data Retrieval in a Subblock Order

Read Control Interface Signals

The read interface uses the same address and data signals previously described; however, read transactions can be identified by the states of the Rd* and Wr* pins, as noted in the address generation section.

Read Interface Timing Diagrams

Cache refills occur using a 4-word subblock ordered sequence. In subblock ordering, the processor delivers the address of the requested word within the block. An external agent must return the block of data using subblock ordering, starting with the addressed word.

In general, a block of data elements (whether bytes, halfwords, words, or doublewords) can be retrieved from storage in a subblock ordered sequence, as shown in Figure 9.6.

Various read operations are shown in the following timing diagrams: Figure 9.7, a single-word back-to-back read cycle; Figure 9.8, a cache line read from a 32-bit port device; Figure 9.9,

[1] Using the subblock ordering shown in Figure 9.6, the word at the target address is retrieved first (Word 2), followed by the remaining word (Word 3) in this doubleword. Next, the doubleword that fills out the quadword is retrieved in the same order as the prior doubleword (in this case, Word 0 is followed by Word 1).

Notes

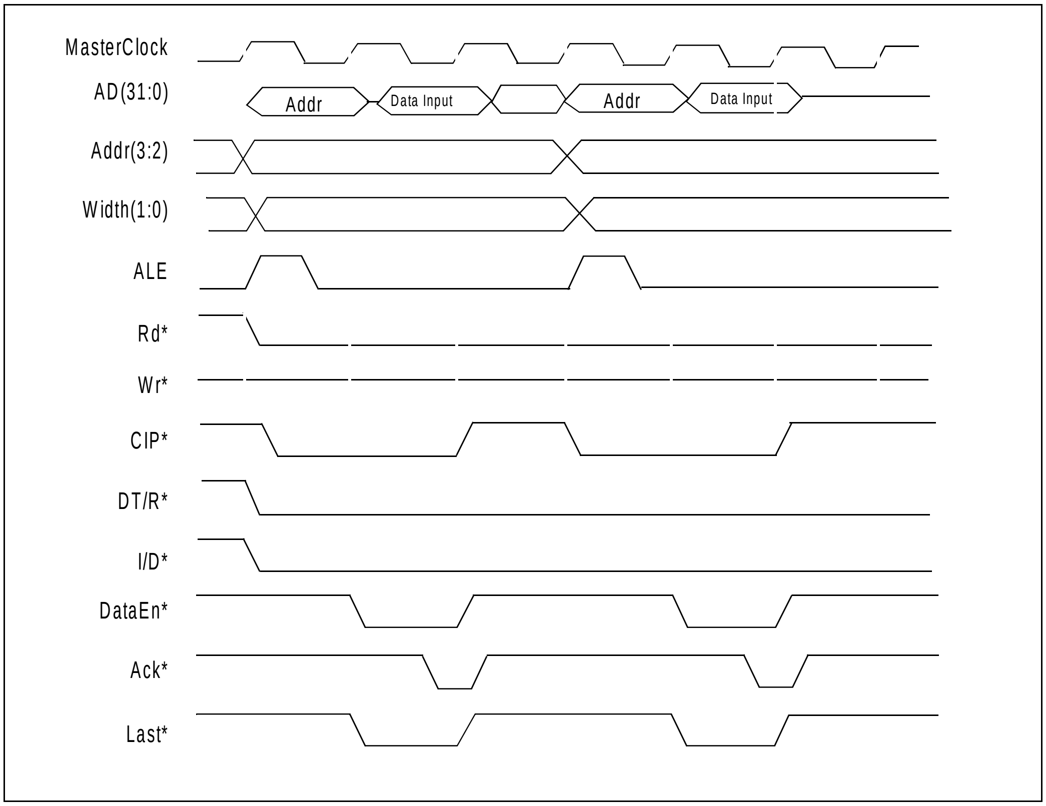


Figure 9.7 Single-Word Back-to-Back Read Cycles

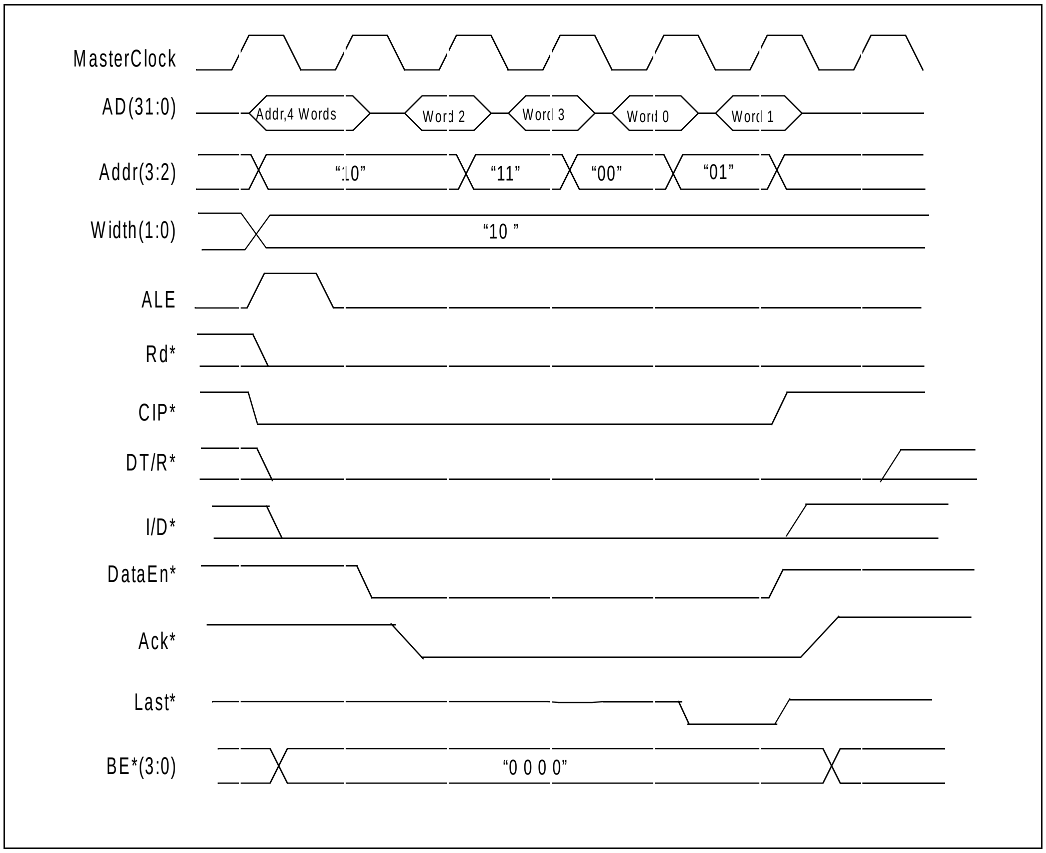


Figure 9.8 Cache Line Read from a 32-bit Port Device

Notes

The timing of a cache line read operation, from a 32-bit port device, is shown in Figure 9.8. The timing of a cache refill using an eight-halfword subblock ordered sequence is shown in Figure 9.9.

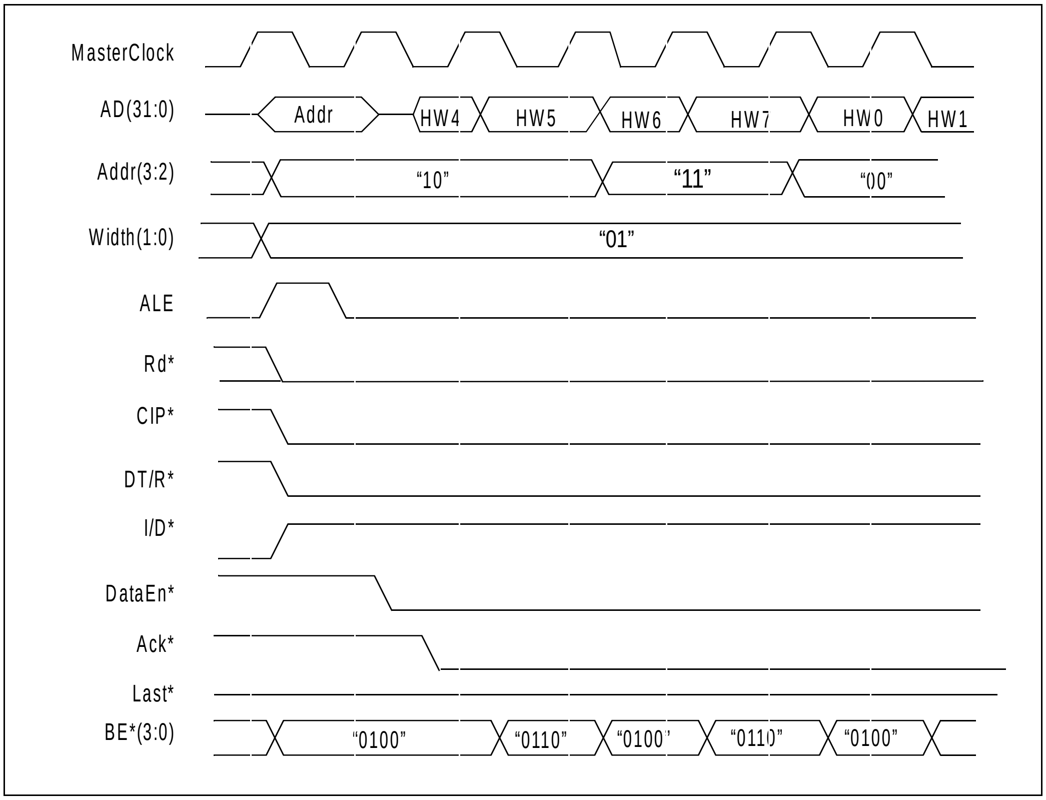


Figure 9.9 Cache Refill Using an Eight-Halfword Subblock Ordered Sequence from 16-bit Port Device (Part 1)

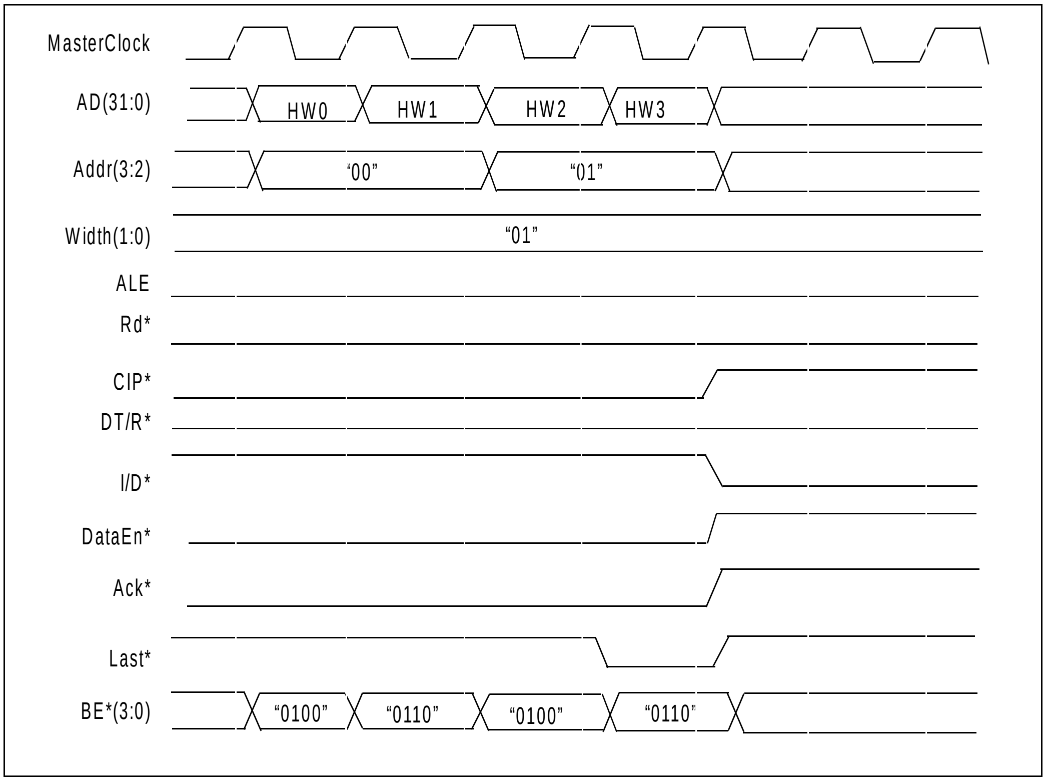


Figure 9.10 Cache Refill Using an Eight-Halfword Subblock Ordered Sequence from 16-bit Port Device (Part 2)

Notes

Write Control Interface

Writes can be single-data (including mini-bursts to narrow ports) or cache-line writebacks. In the case of cache line writebacks, data is written using subblock ordering, beginning from word 0 of the line. Figure 9.11, Figure 9.12, Figure 9.13, Figure 9.15 and Figure 9.16 illustrate the timings of various write control operations.

Timing Diagrams for Write Interface Operations

Throughout the following tables and figures, pin names followed by an asterisk are active when low.

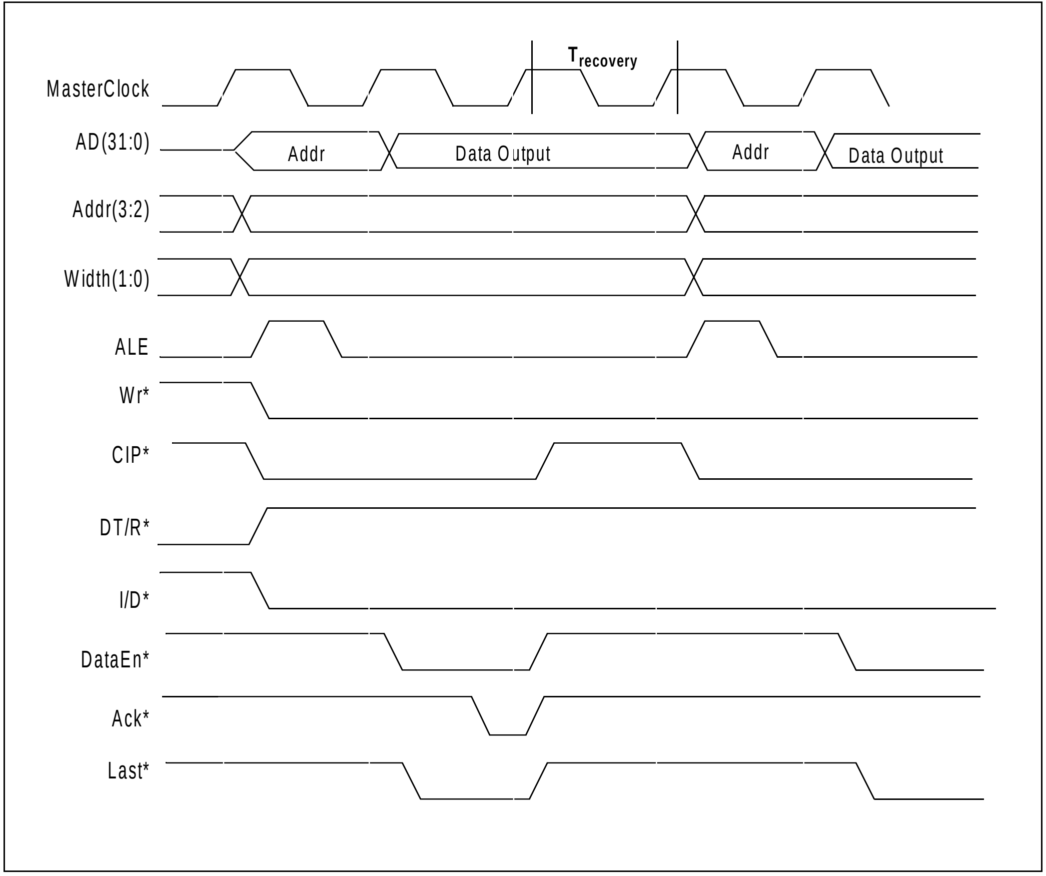


Figure 9.11 Single-Word Back-to-Back Write Cycles

Notes

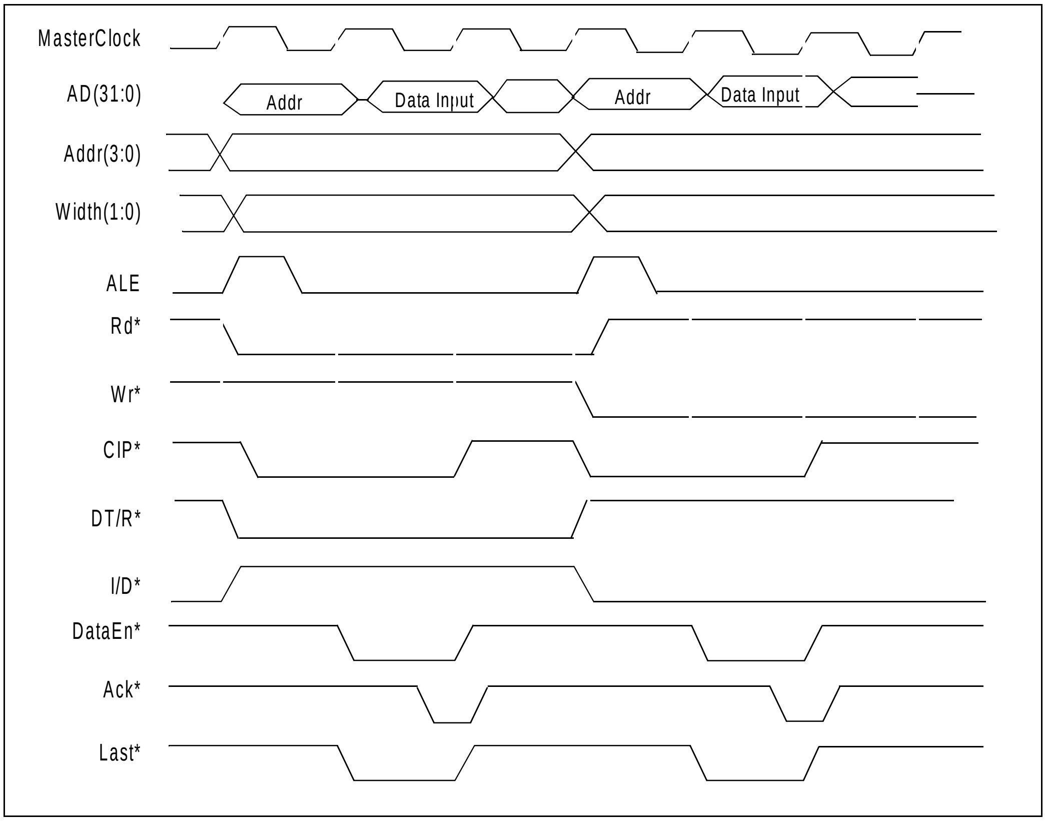


Figure 9.12 Single-Word Back-to-Back Read, Followed by a Write Cycle

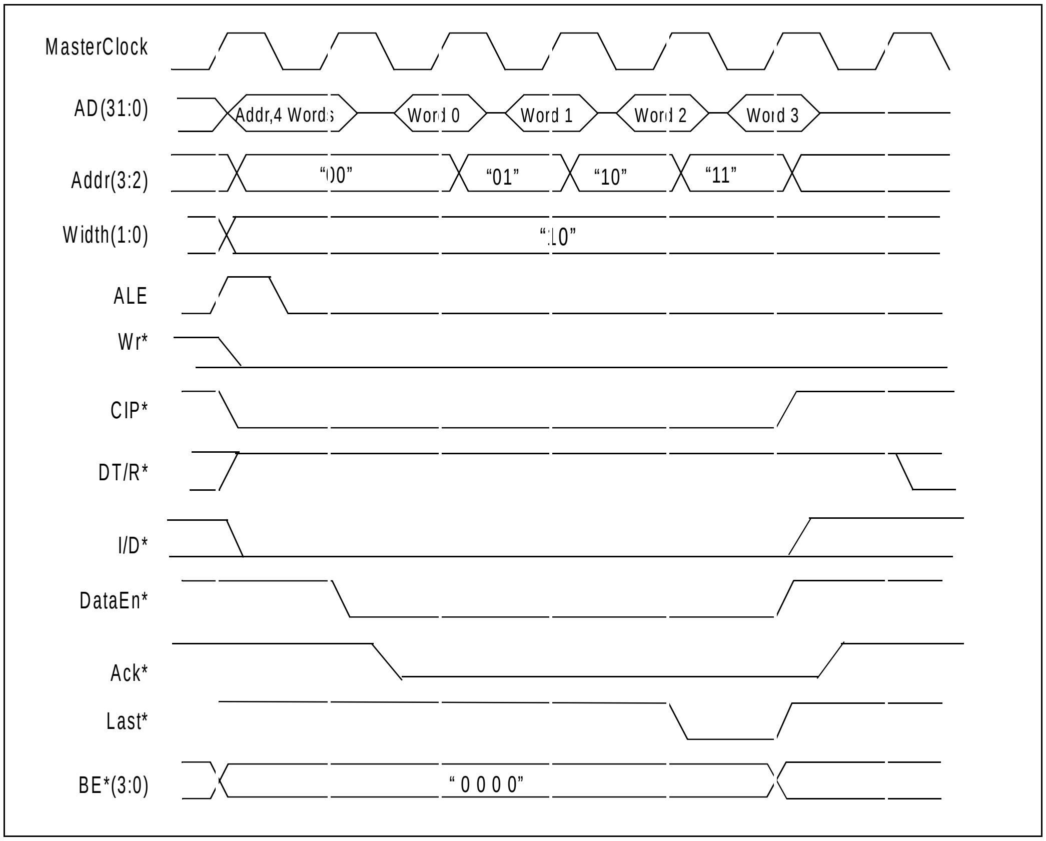


Figure 9.13 Burst Write to a 32-bit Port

Notes

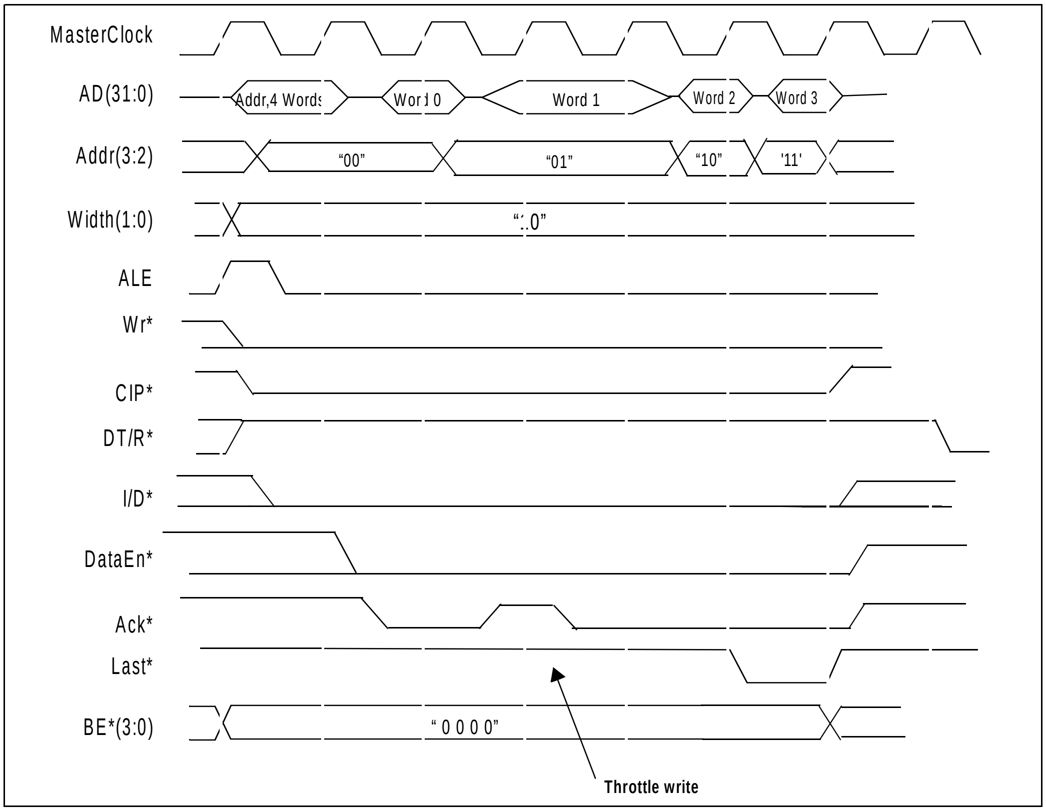


Figure 9.14 Write Throttle Timing Diagram

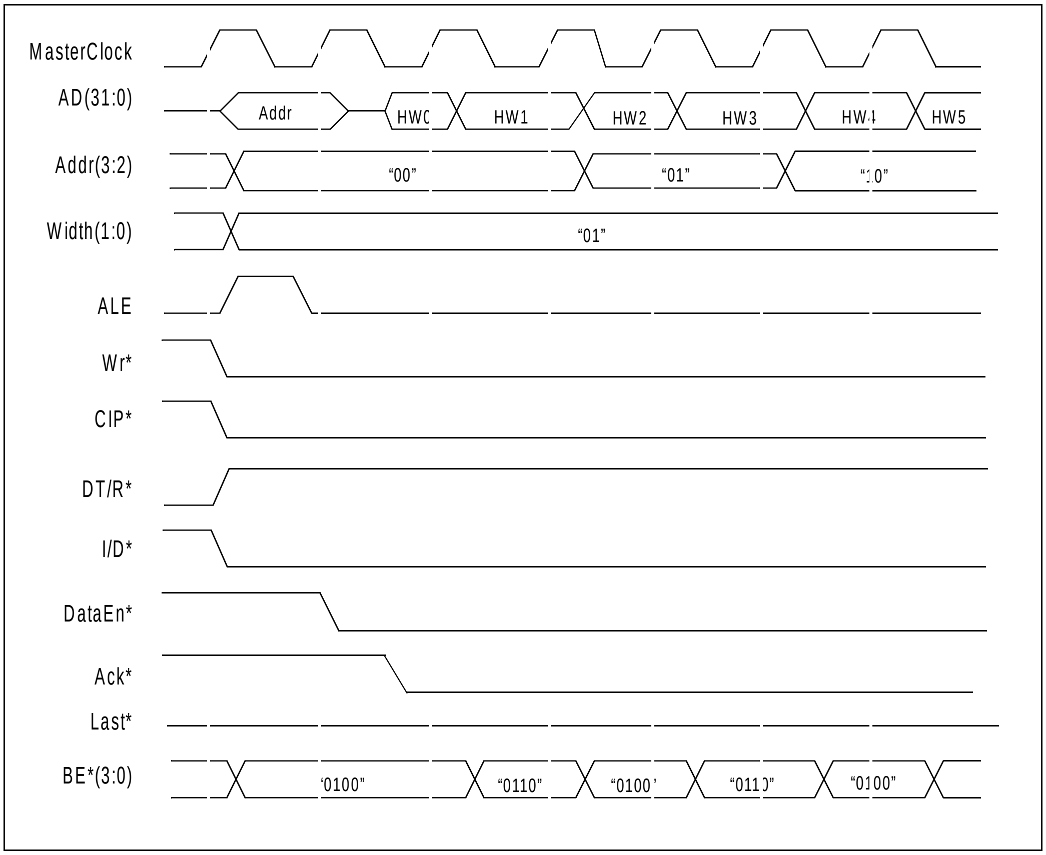


Figure 9.15 Eight HalfWords Sequential Ordering Write (Part 1)

Notes

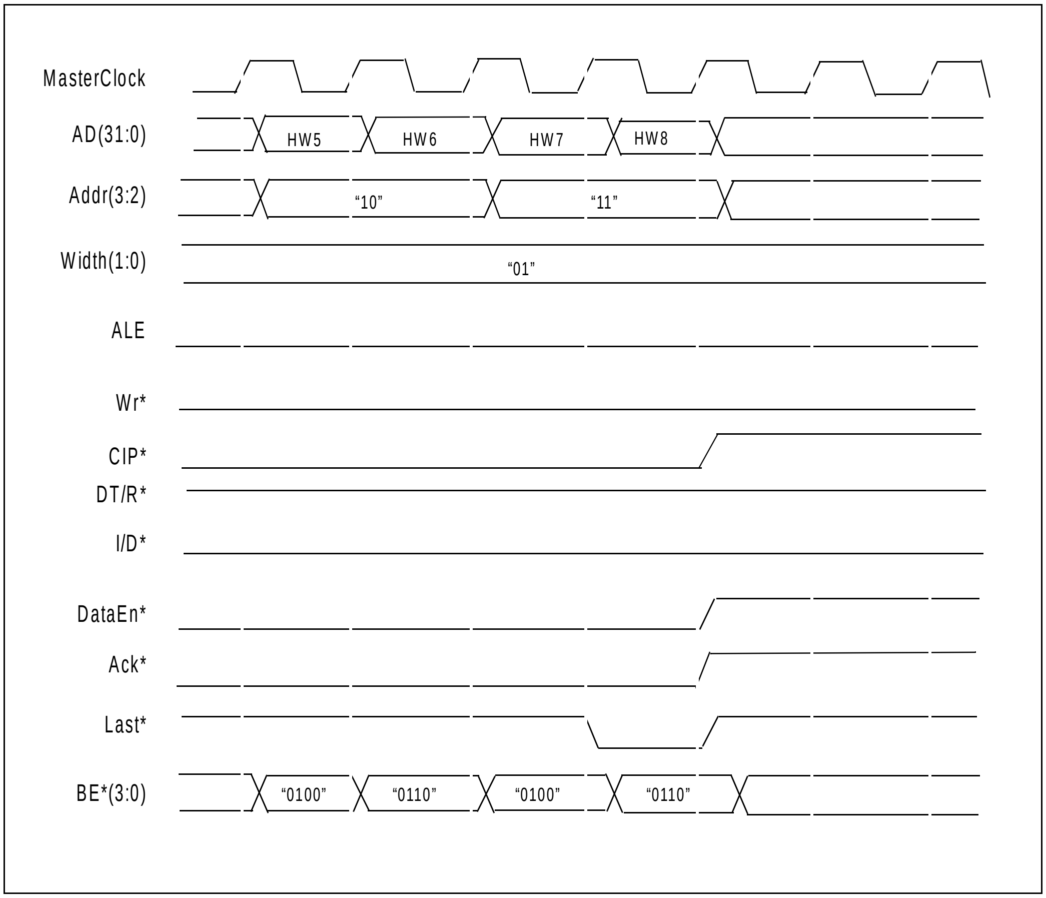


Figure 9.16 Eight HalfWords Sequential Ordering Write (Part 2)

Notes

Bus Retry

During the first data phase, external devices can force the RC32364 bus to terminate the current bus transaction by asserting Retry* (without asserting Ack*). Once Ack* is sampled and the first data has been accepted, Retry* will be ignored.

If BusReq* was also sampled asserted, the processor relinquishes bus ownership by asserting BusGnt*. If both Retry* and Ack* are sampled asserted, RC32364 will ignore the Retry* and complete the current transaction.

If Retry* is sampled asserted while both Ack* and BusReq* are de-asserted, then the RC32364 will terminate the current transaction and immediately retry the transaction.

The timing of a bus retry with a pending BusReq* is illustrated in Figure 9.17. The timing of a bus retry without a pending BusReq* is shown in Figure 9.18. Figure 9.19 illustrates the timing of an operation with both the Retry***** and Ack***** signals asserted.

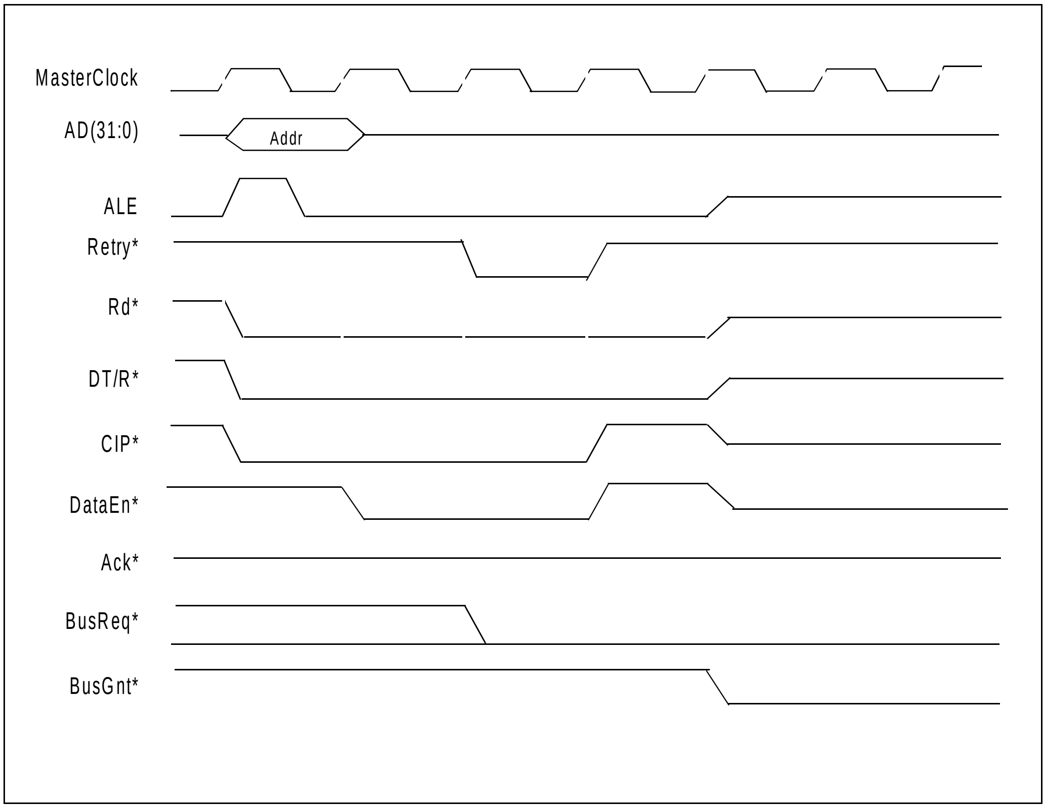


Figure 9.17 Bus Retry with Pending BusReq*

Notes

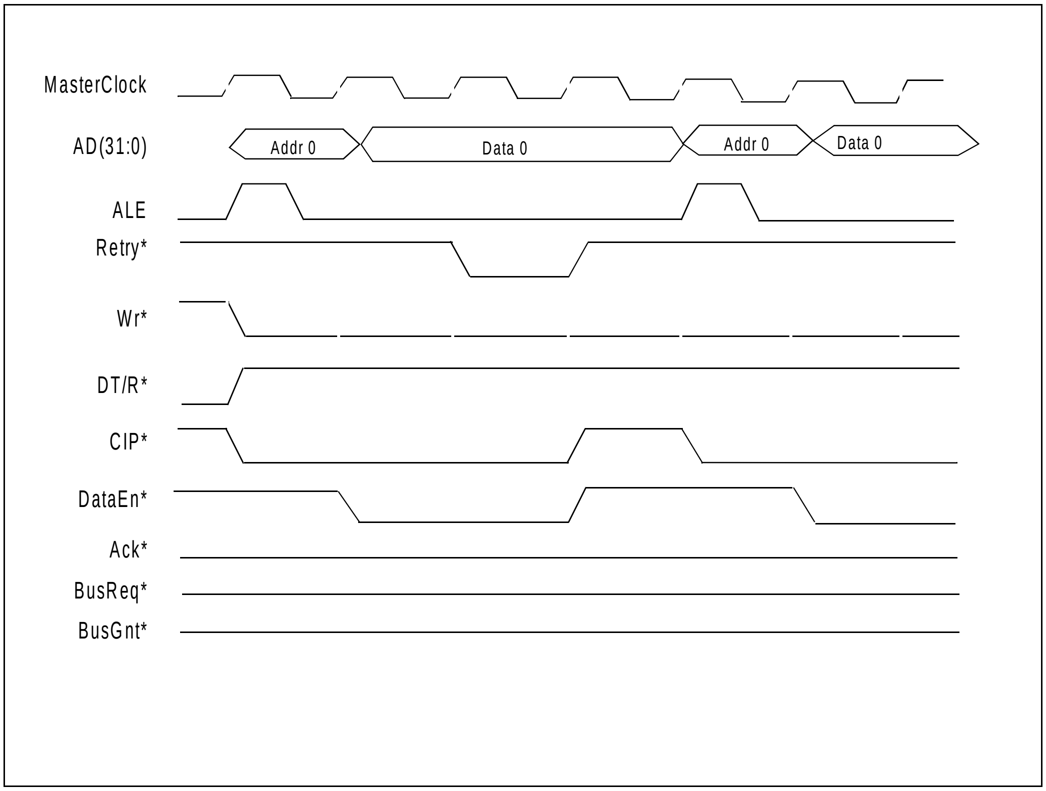


Figure 9.18 Bus Retry without a Pending BusReq*

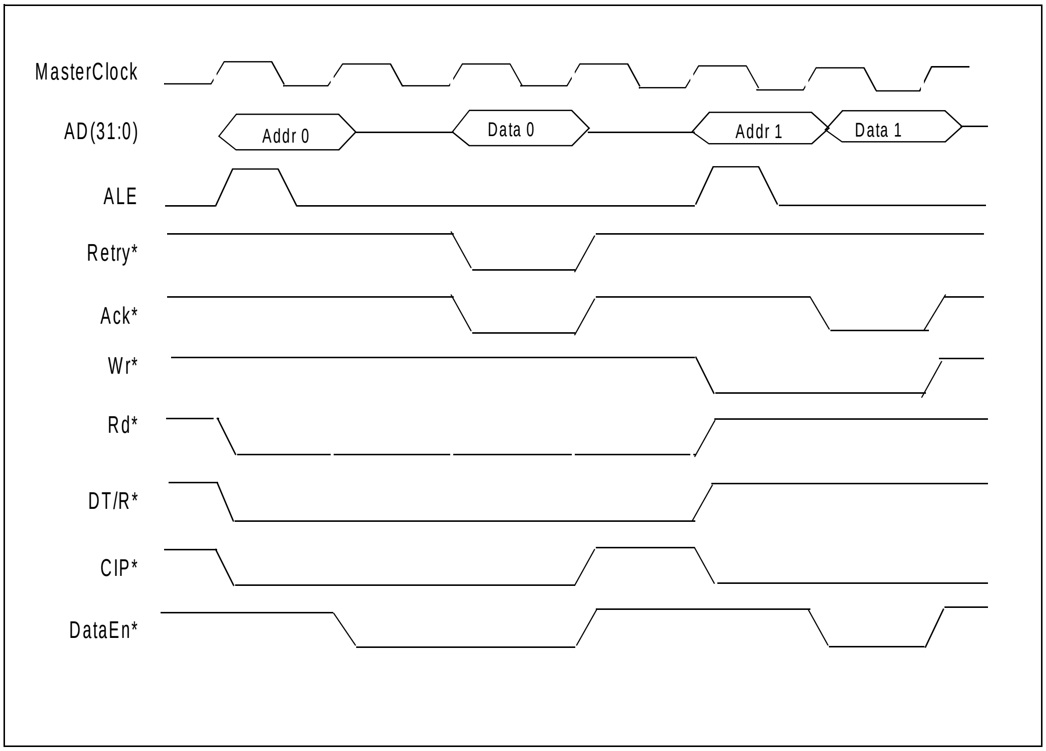


Figure 9.19 Assertion of Retry* along with Ack* Asserted

Notes

Bus Error

If the RC32364 tries to access a memory region that does not have any device connect to it, or the external controller finds a parity error, BusErr* can be asserted, which forces the RC32364 into taking the bus error exception. BusErr* is ignored after the acceptance of the first data. If BusErr* and Ack* are both sampled asserted in the first data phase, Ack* will be ignored, and the bus error exception will be taken.

The timing of the BusErr* operation is illustrated in Figure 9.20.

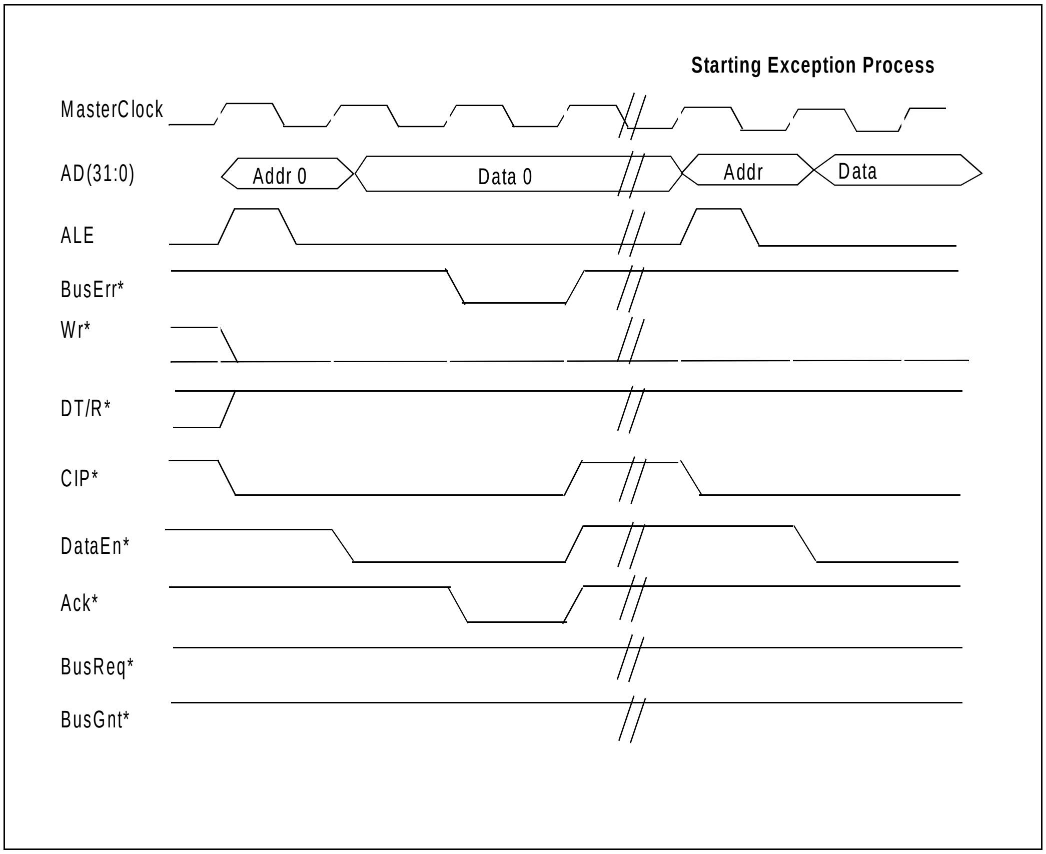


Figure 9.20 Bus Error

DMA Mastership Interface

The DMA interface uses a simple two signal protocol, to allow an external agent mastership of the system bus. Internal CPU Logic synchronizes the external interface to the internal arbiter unit. This insures that no conflicts occur between the internal synchronous requesters (read and write engines) and the external asynchronous (DMA) requester. Table 9.14 lists the pin names and descriptions for the two DMA interface signals. Both signals are active when low.

Pin	Signal	Description
BusReq*	Bus Request (I)	This active low signal is an input to the processor, used to request mastership of the external interface bus. Mastership is granted according to the assertion of this input, and taken back based on its negation.
BusGnt*	Bus Grant (O)	This active low signal is an output from the processor. It is used to indicate that the CPU has relinquished mastership of the external interface bus. BusGnt* goes low initially for at least 2 clocks to indicate that the CPU has relinquished mastership of the external interface bus. After going low, BusGnt* returns high either when the CPU makes an internal request for the bus or after BusReq* is de-asserted.

Table 9.14 DMA Interface Pin Descriptions

Notes

DMA Mastership Initiation

Figure 9.21 shows the beginning of a DMA cycle. Note that if BusReq* is asserted while the processor was performing a read or write operation, BusGnt* would be delayed until the next bus slot after the read or write operation is completed.

To initiate DMA, the processor must detect the assertion of BusReq* with proper set-up time to MasterClock. Once BusReq* is detected, and the bus is free, the processor will grant control to the requesting agent by asserting its BusGnt* output, and tri-stating its output drivers. The bus will remain under the control of the external master until it negates BusReq*, indicating that the processor is once again the bus master. BusReq* needs to assert and de-assert for at least 2 cycles.

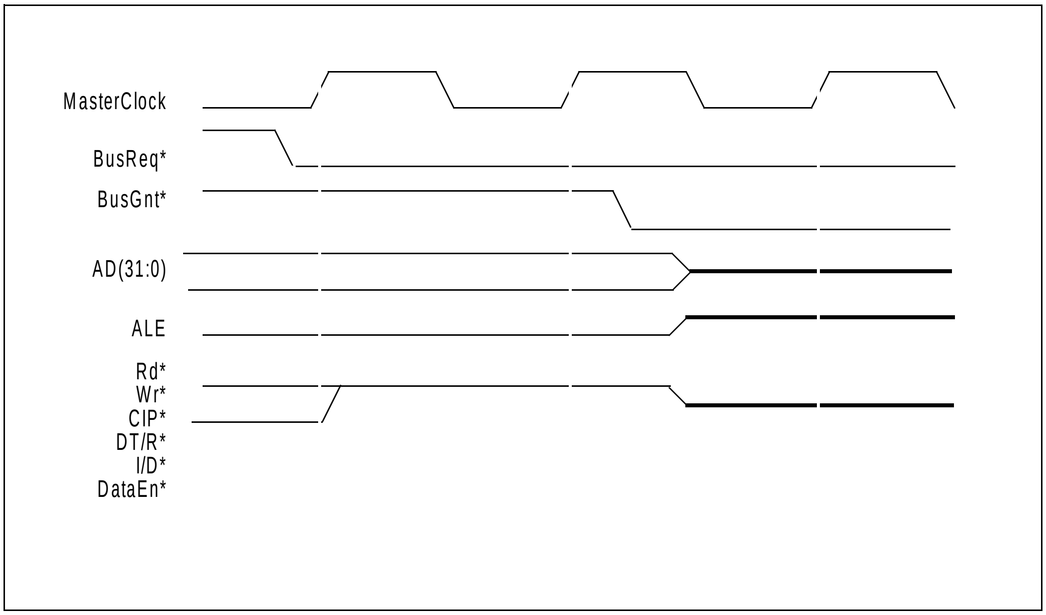


Figure 9.21 Bus Grant and Start of DMA Transaction

Notes

Relinquishing Mastership Back to the CPU

The next rising edge of MasterClock—after the negation of BusReq* is sampled—may actually be the beginning of a processor read or write operation. To terminate DMA, the external master must negate the processor BusReq* input.

Once this is detected (with proper setup and hold time), the processor will negate its BusGnt* output on the next rising edge of MasterClock, if it hasn't already done so. It will also re-enable its output drivers. Thus, the external agent must disable its output drivers by this clock edge, to avoid bus conflict.

Figure 9.22 shows the end of a DMA cycle.

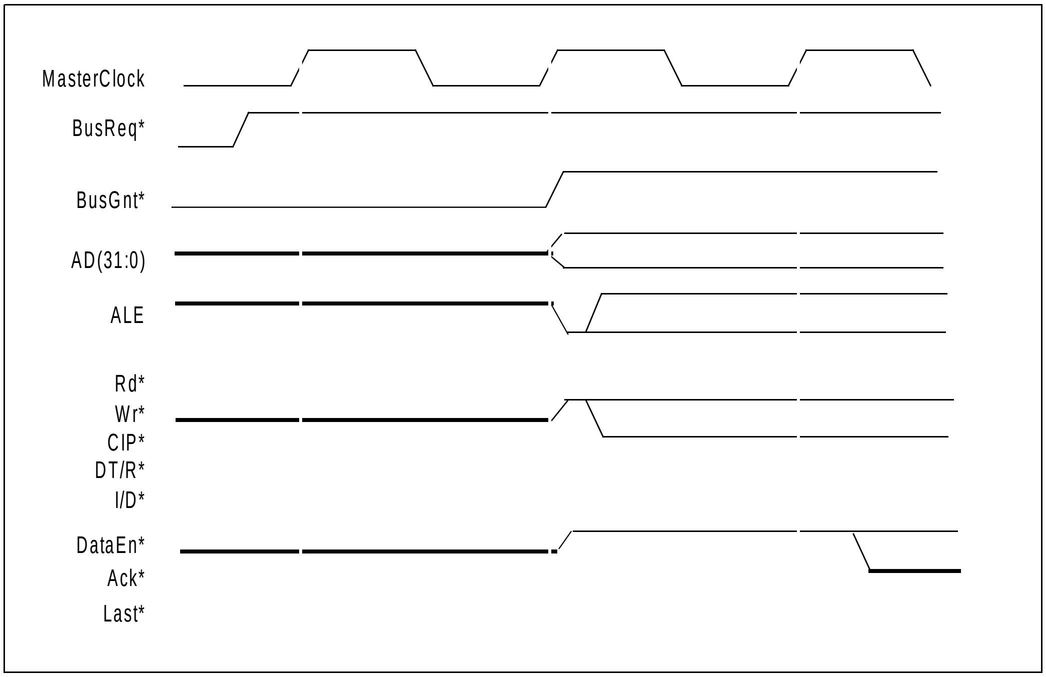


Figure 9.22 Regaining Bus Mastership

CPU Initiated Bus Grant De-Assertion

Figure 9.23 indicates the middle of a DMA cycle, where the processor generates an internal request for the external bus. If BusGnt* has been low for at least 2 clock periods, and the CPU has a pending external request, then on the next rising edge of MasterClock, BusGnt* will be de-asserted. This signals to the external agent that the CPU could benefit from bus usage. However, the external agent remains in control of the bus until it negates BusReq*.

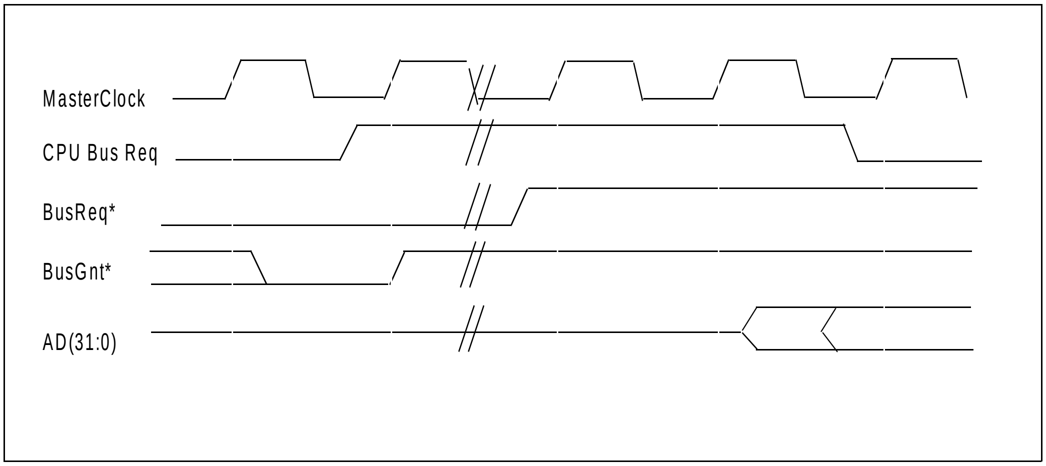


Figure 9.23 DMA Protocol BusGNT* De-assertion

RC32364 Enhancements to MIPS 32 ISA

Notes

Introduction

The RC32300 execution unit implements an extended version of the MIPS 32 ISA. 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 RC32364 follows. General instruction set information can be found in the *IDT MIPS Microprocessor Family Software Reference Manual*.

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 Table 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.

The load operations in the RC32364 are blocking, which means that prefetch load operations will also be blocking.

31 26	25 21	20 16	15 0
PREF 110011	base	hint	offset
6	5	5	16

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

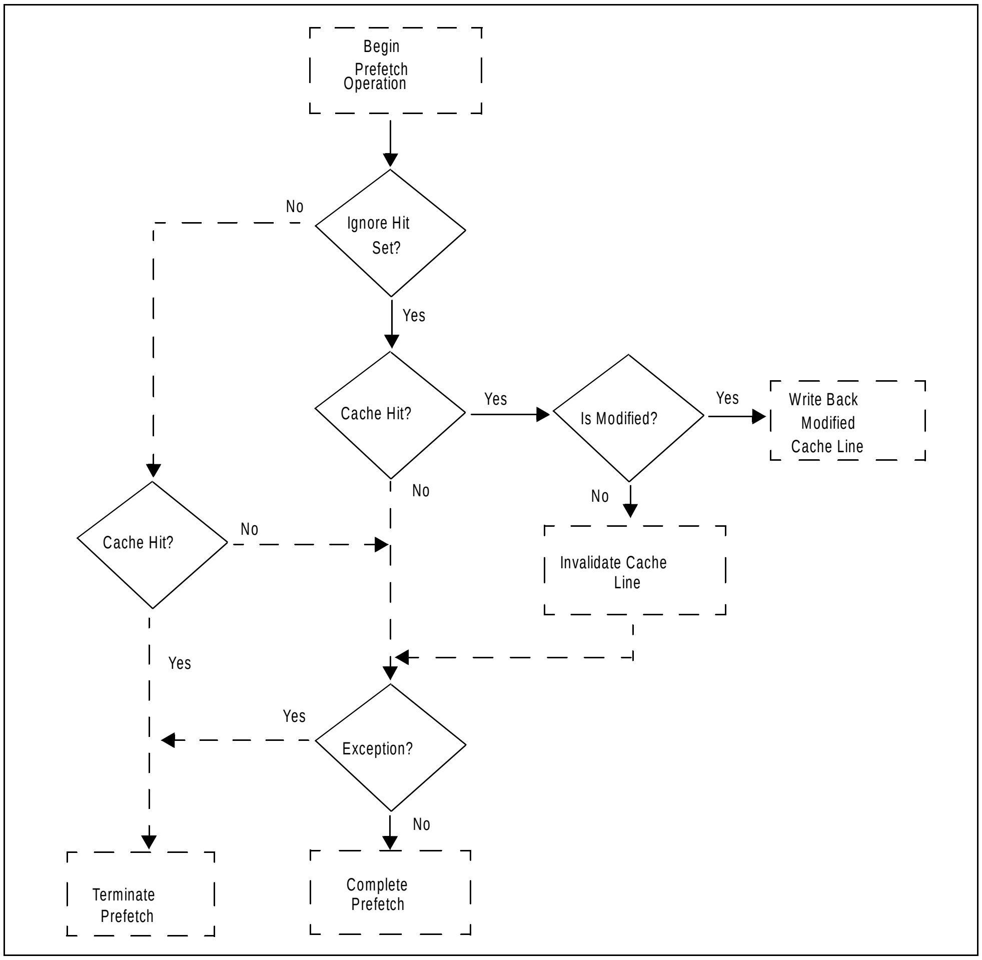


Figure A.2 Flowchart for Prefetch Operation

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

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 RC32364 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

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

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] ≠ 0 then GPR[rd] <-- GPR[rs]
```

Exception: Reserved Instruction.

Move Conditional on Zero

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

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.

Notes

Instructions for DSP Support

The RC32364 adds new instructions to the MIPS 32 ISA, intended to enhance the performance of certain types of DSP algorithms.

Specifically, enhancements in the multiplier have been added to allow fast fused multiply-adds and multiply-subtracts. In addition, RC32364 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).

Multiply Add

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	0 00000	0 00000	MAD 000000
6	5	5	5	5	6

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 <-- temp$_{31..0}$
       HI <-- temp$_{63..32}$
```

Exception: None

Multiply Add Unsigned

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	0 00000	0 00000	MADU 000001
6	5	5	5	5	6

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 <-- temp$_{31..0}$
       HI <-- temp$_{63..32}$
```

Exception: None

Notes

Multiply Subtract

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	0 00000	0 00000	MSUB 000100
6	5	5	5	5	6

Format: MSUB 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 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 <-- $temp_{31..0}$
       HI <-- $temp_{63..32}$
```

Exception: None

Multiply Subtract Unsigned

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL2 011100	rs	rt	0 00000	0 00000	MSUBU 000101
6	5	5	5	5	6

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 <-- $temp_{31..0}$
       HI <-- $temp_{63..32}$
```

Exception: None

Notes

Count Leading Zeros

31 26	25 21	20 16	15 6	5 0
SPECIAL2 011100	rs	rt	0 0000000000	CLZ 100000
6	5	5	10	6

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

Count Leading Ones

31 26	25 21	20 16	15 6	5 0
SPECIAL2 011100	rs	rt	0 0000000000	CLO 100001
6	5	5	10	6

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

RC32364 Opcode Map

Notes

Opcode

	28..26							
31..29	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

	2..0							
5..3	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

	2..0							
5..3	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

23..21 25,24	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

18..16 20..19	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

2..0 5..3	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

18..16 20..19	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 RC32364, both an I-cache and D-cache access can occur simultaneously; cache accesses take one processor clock to complete.

Information specific to the RC32364's cache organization and operation is provided in Chapter 6 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.*

Notes

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.
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

Notes

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

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 AD 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 AD bus and ending when the last word of data appears on the AD bus, inclusive.

Notes

RC32364 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 RC32364 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 RISCore32300TM core is a static design, and products based on this core, such as the RC32364, offer various power reduction modes. In addition, the RISCore32300TM 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.

31 26	25	24 6	5 0
COP0 0 1 0 0 0 0	CO 1	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	WAIT 1 0 0 0 0 0
6	1	19	6

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
```

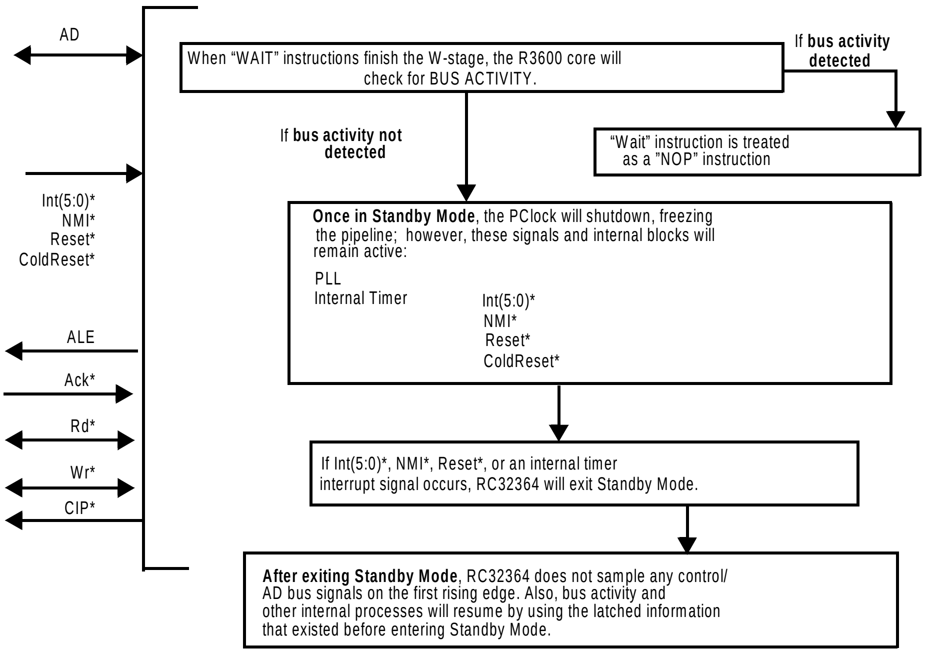
Exceptions: Coprocessor unusable exception.

Notes

Entering Standby Mode

To enter standby mode, first execute the WAIT instruction. When the WAIT instruction finishes the W pipe-stage, if the **AD** bus is currently idle, the internal clocks will shut down, thus freezing the pipeline. The PLL, internal timer, some of the input pin clocks (**Int[5:0]***, **NMI***, **Reset*** and **ColdReset***) will continue to run.

If the conditions are not correct when the WAIT instruction finishes the W pipe-stage (such as the **AD** 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—will cause the CPU to exit standby mode. Figure D.1 illustrates the flow of the Standby Mode Operation.


AD
Int(5:0)*
NMI*
Reset*
ColdReset*
ALE
Ack*
Rd*
Wr*
CIP*
When "WAIT" instructions finish the W-stage, the R3600 core will check for BUS ACTIVITY.
If bus activity detected
"Wait" instruction is treated as a "NOP" instruction
If bus activity not detected
Once in Standby Mode, the PClock will shutdown, freezing the pipeline; however, these signals and internal blocks will remain active:
PLL
Internal Timer
Int(5:0)*
NMI*
Reset*
ColdReset*
If Int(5:0)*, NMI*, Reset*, or an internal timer interrupt signal occurs, RC32364 will exit Standby Mode.
After exiting Standby Mode, RC32364 does not sample any control/ AD bus signals on the first rising edge. Also, bus activity and other internal processes will resume by using the latched information that existed before entering Standby Mode.

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 RC32364 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 more information.

List of Hazards

RC32364 CP0 hazards are as follows:

- *An mtc0 followed by an mfc0 is undefined. A one instruction delay between mtc0 and mfc0 is needed for proper operation.*
- *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.*

Example:

```
MTC0 CO_STATUS, R5

NOP

NOP

ERET
```

Notes

Integer Multiply Scheduling

Notes

Introduction

Integer multiply performance is substantially enhanced in the RC32364. The RC32364 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 RC32364 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	$-2^{15} \le rt \le 2^{15}-1$	3	2	0
MULT, MAD	$rt < -2^{15}$ or $rt > 2^{15}-1$	4	3	0
MULTU, MADU	$0 \le rt \le 2^{16}-1$	3	2	0
MULTU, MADU	$rt > 2^{16}-1$	4	3	0
MUL	$-2^{15} \le rt \le 2^{15}-1$	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.

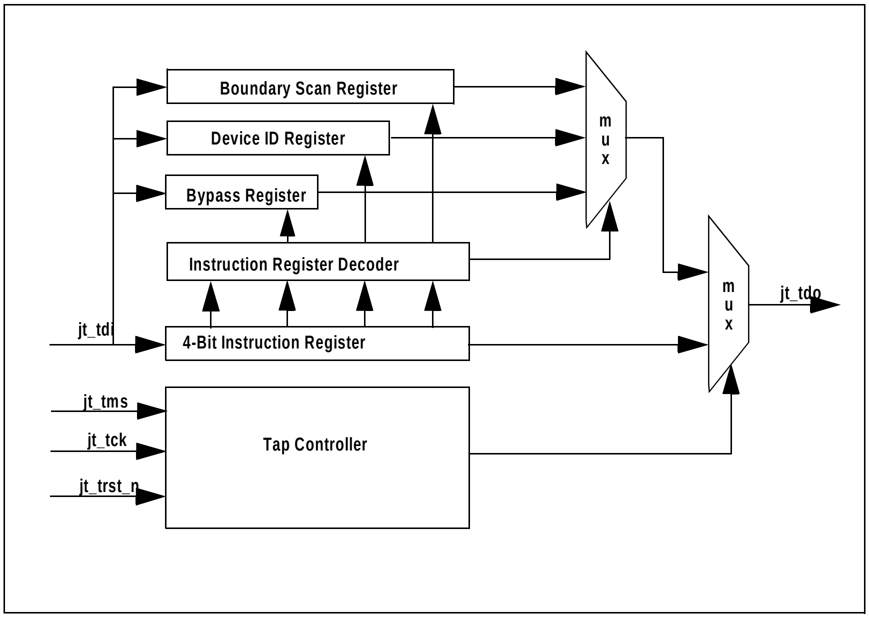
On the RC32364, the result is available immediately and there is no slip.

EJTAG (In-circuit Emulator) Interface

Notes

Introduction

The RC32364 TAP Controller is used to provide access to the EJTAG interface on the CPU core.



On-chip support for low-cost in-circuit emulation (ICE) equipment is featured on the RC32364. The RC32300 CPU core on the RC32364 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 maximum speed of the DCLK that the EJTAG probe can sustain.*
- *Debug breaks can be initiated by the EJTAG Probe via a JTAG pin (*TDI/DINT**).*
- *PC Trace information is provided by additional status pins and the processor clock.*

Notes

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.). 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 G.1, and the simplified block diagram is shown in Figure G.2.

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*

Block Diagrams

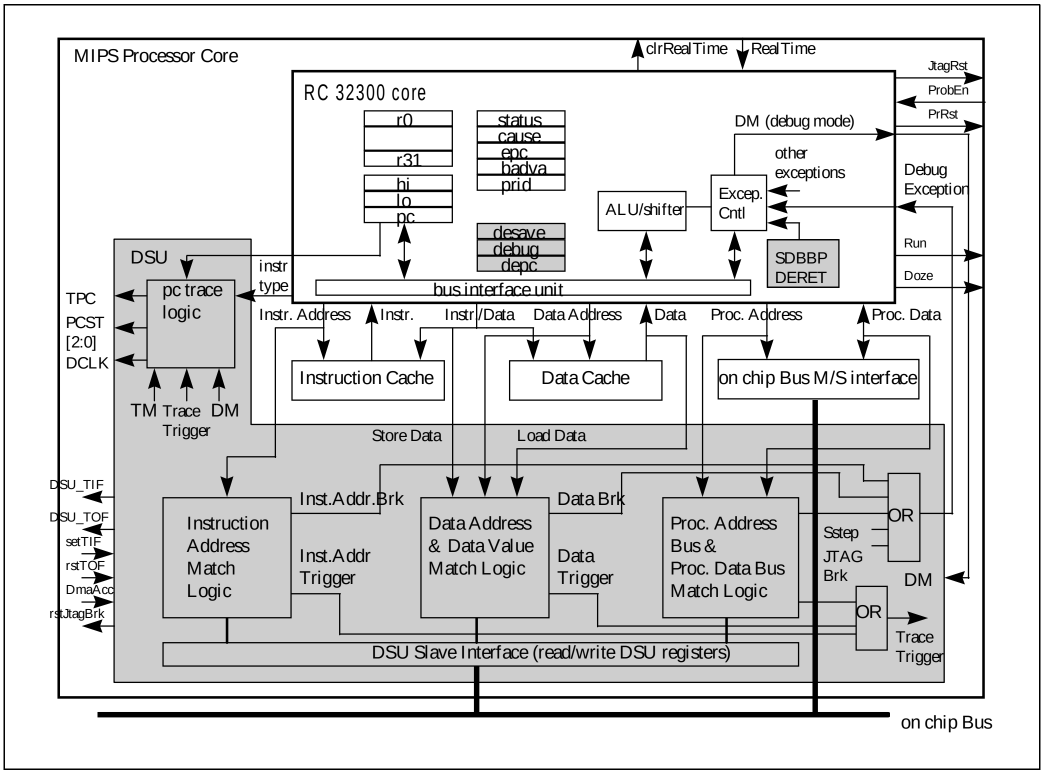


Figure G.1 Block Diagram

Notes

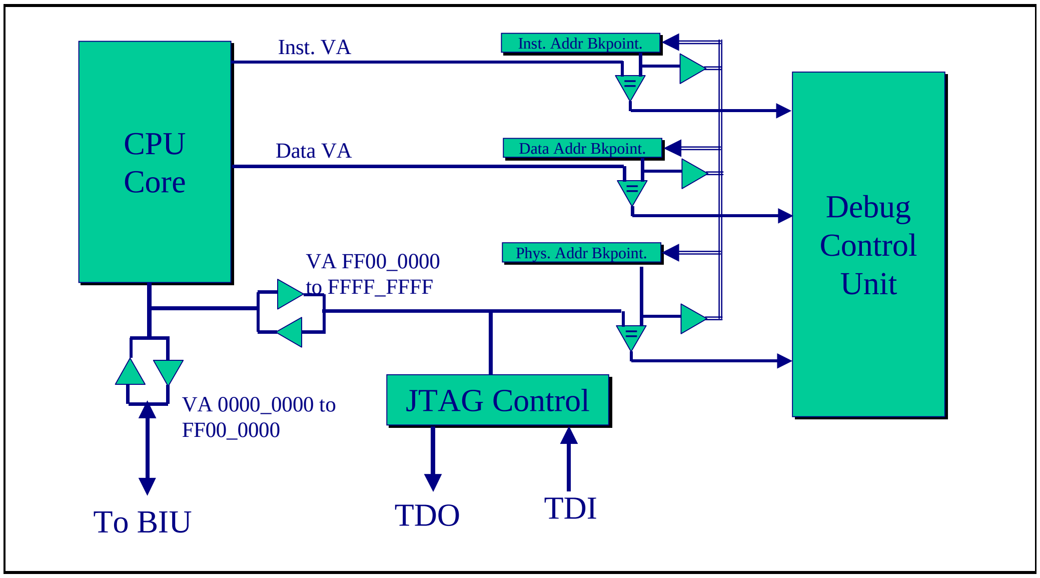


Figure G.2 Simplified EJTAG Block Diagram

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. Additional status pins (for Run-time and Real-time data collection) along with 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 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 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 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.

Notes

EJTAG Interface

The EJTAG interface consists of the standard JTAG signals (i.e. TCK, TMS, TDI, TDO, TRST*), extended with extra signals that provide real time program counter output. A description of the EJTAG pins is shown in Table G.1.

Pins	Input/ Output	Description
TCK	I	Test Clock Input Input clock used to shift data into or out of the JTAG Instruction or Data register. The TCK clock may be independent of the processor clock.
TDI/DINT*	I	Test Data Input / Debug Interrupt ***PC Trace mode off*:** serial input data (TDI) is shifted into the JTAG Instruction register or Data register on the rising edge of the TCK clock, depending on the TAP controller state. ***PC Trace mode on:*** an active LOW level at this input (DINT*) is used as interrupt to switch the PC Trace mode off (see DCLK description). This signal is sampled at the TCK positive edge or asynchronous to TCK.
TDO/TPC	O	Test Data Output / Target PC Output ***PC Trace mode off***: serial output data is shifted from the JTAG Instruction or Data register to the TDO/TPC pin at the falling edge of the TCK clock. When no data is shifted out, the TDO/TPC is tri-stated. **PC Trace mode on:** this pin provides non-sequential program counter output (TPC) at the processor clock (DCLK).
TMS	I	Test Mode Select Input The TMS input signal is decoded by the TAP controller to control test operation. TMS is sampled on the rising edge of TCK.
TRST*	I	Test Reset Input The TRST* pin is an active-low signal for asynchronous reset of the EJTAG Module, independent of the processor logic. Requires an external pull-down on the board.
RST*	I	Reset Input for Board The RST* pin is an active-low signal for asynchronous reset of the entire target ***board***. It is not implemented directly in the EJTAG module on the RC32364, but it is rather a pin of the EJTAG connector that goes to the target board.
DCLK	O	Processor Clock During PC Trace mode this signal is used to capture address and data from the TDO/TPC pin at the processor clock speed. This pin may not be needed if there is already a clock output pin on the processor. The clock at the DCLK pin is the CPU clock. DCLK is disabled (tri-stated or made 0) via ClkEn bit in EJTAG_Control_register[0] or when no probe is present (ProbEn=0 in EJTAG_Control_register[15]). The DCLK is at the same frequency as the CPU pipeline frequency. It has a limit of 100MHz.
PCST [2:0]	I/O	PC Trace Status Information More information on the PC Trace Status Information is in the PC Trace section.
DebugBoot	I	Debug Boot The Debug Boot 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 will also set the JtagBrk bit in the EJTAG_Control_Register[12].
PCST[4:3]	I/O	Reserved

Table G.1 EJTAG Pins

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.

Notes

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.

Figure G.3 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 TDI/DINT* and TDO/TPC function as TDI input and 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 TMS signal. In PC Trace mode, Program counter trace information is output via additional status pins in conjunction with the JTAG pins TDI/DINT* and TDO/TPC. These pins now function as DINT* input and TPC output. Non-sequential program counter data is available at the TDO/TPC pin clocked out at the processor speed using the DCLK pin. The type of execution is available as status at the 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 TDI/DINT* 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

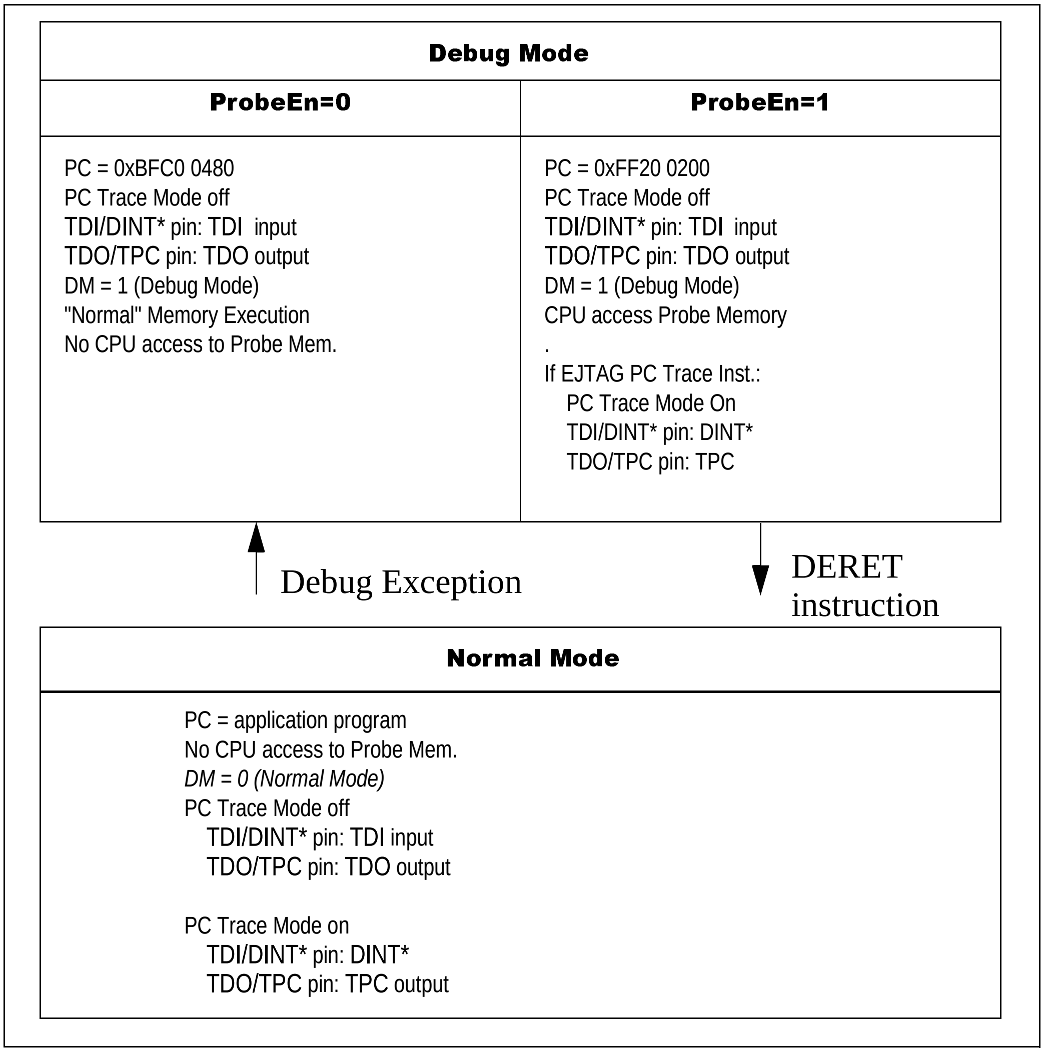


Figure G.3 RC32364 Debug Operating Modes

Test Interface and Boundary-Scan Architecture

The IEEE 1149.1 architecture is shown in the shaded part of Figure G.1. 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 (TDI) pin and primary Test Data Output (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 (TCK) and Test Mode Select (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 TCK input, which responds to the TMS input as shown in the state diagram in Figure G.4. The IEEE 1149.1 test bus uses both clock edges of TCK. TMS and TDI are sampled on the rising edge of TCK, while TDO changes on the fall

The state diagram for the TAP Controller is shown in Figure G.4.

Notes

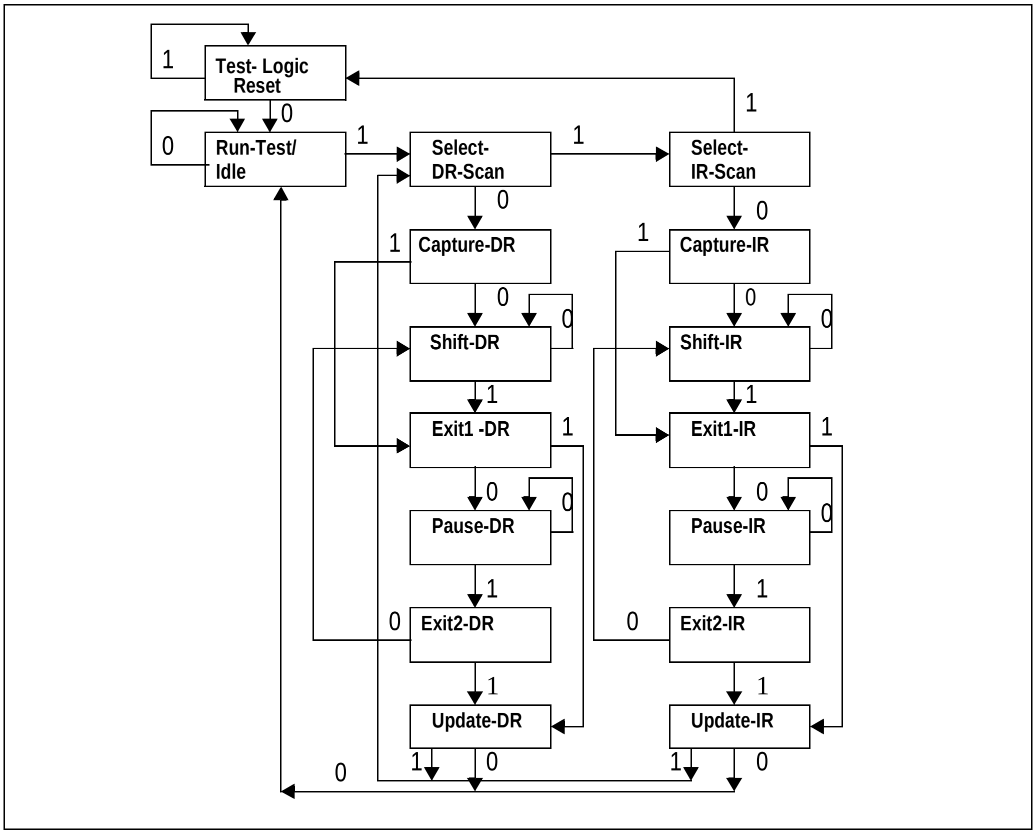


Figure G.4 TAP Controller State Diagram

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 TMS is set high: the Test-Logic-Reset state. This means that a reset of the test logic can be achieved within five TCK(s) or less by setting the 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 TMS high and applying five or more 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 TMS and 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 G.4.

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 KOA.

TAP Controller State Assignments

All state transitions within the TAP controller occur at the rising edge of the TCLK pulse and—depending on the 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 TDI and 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 TDO output.*
- *Exit1-DR*
  - *This is a temporary controller state. If 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 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 TDI and TDO to be temporarily halted.*
- *Exit2-DR*
  - *This is a temporary controller state. If 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 TDI and 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 TDO output.*

Notes

- *Exit1-IR*
  - *This is a temporary controller state. While in this state, if 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 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 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 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).

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 G.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 implemented in the RC32364, 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 implemented in the RC32364, this value is 0x0	R	impl. dep.

Table G.2 CPU Core Device Identification Register

.

Version	Part Number	Vendor ID	LSB
0000	0000|0000|0010|0110	0000|0110|011	1

Figure G.5 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 RC32364.

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 RC32364**	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 G.3 Implementation Register (Part 1 of 2)

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 G.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 RC32364 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 TDO/TPC pin to the EJTAG Probe. Depending on the direction of the access, data is shifted into the TDI pin (processor read) or shifted out of the TDO/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 RC32364 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 TDO/TPC pin to the EJTAG Probe for a Processor Write action; for a Processor Read action 32 bits of data is shifted into the TDI /DINT* 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 G.6 and Figure G.7.

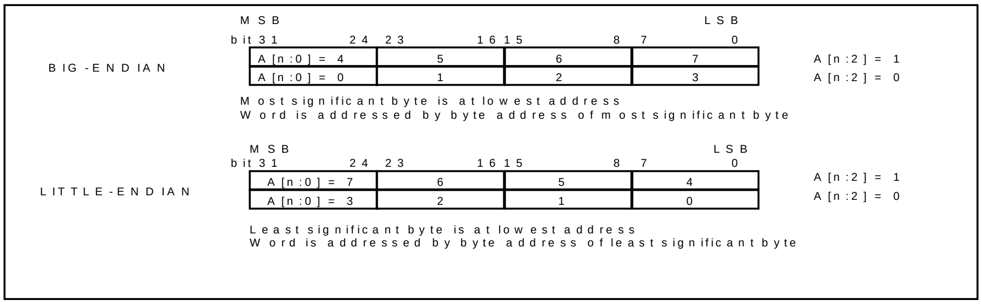

BIG-ENDIAN
MSB
LSB
bit 31 24 23 16 15 8 7 0
A[n:0] = 4
5
6
7
A[n:0] = 0
1
2
3
A[n:2] = 1
A[n:2] = 0
Most significant byte is at lowest address
Word is addressed by byte address of most significant byte
LITTLE-ENDIAN
MSB
LSB
bit 31 24 23 16 15 8 7 0
A[n:0] = 7
6
5
4
A[n:0] = 3
2
1
0
A[n:2] = 1
A[n:2] = 0
Least significant byte is at lowest address
Word is addressed by byte address of least significant byte

Figure G.6 Byte Organization in a 32-bit EJTAG Data Register

Notes

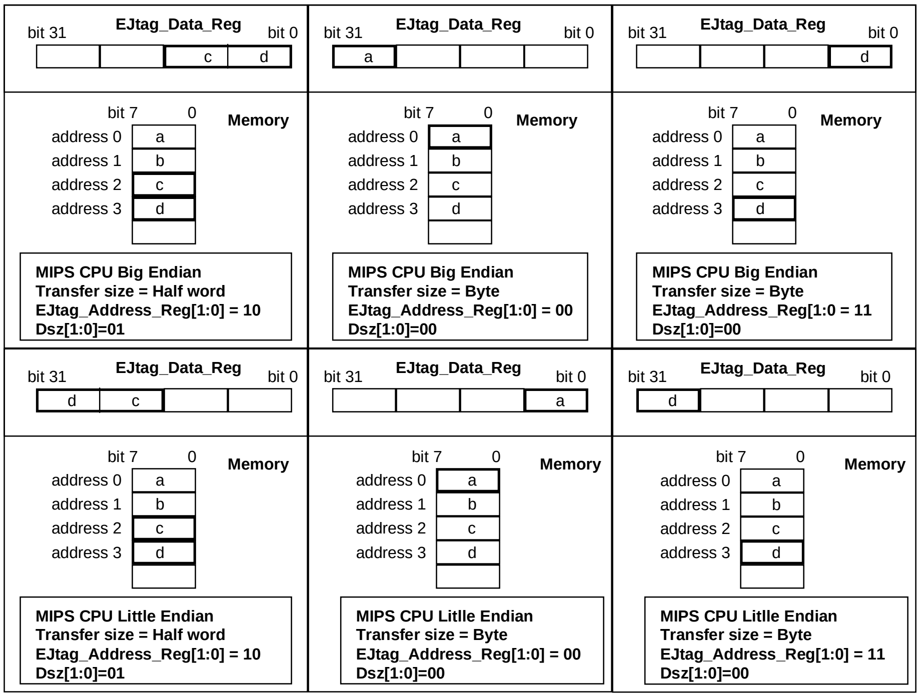


Figure G.7 Examples of Byte Organization in a 32-bit EJTAG Data Register

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 G.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 G.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 TDI/DINT* 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 G.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 RC32364.	R	0
31..26	reserved	**reserved**	R	0

Table G.4 EJTAG_Control_Register (Part 3 of 3)

Figure G.8 shows examples of the Sync Operation.

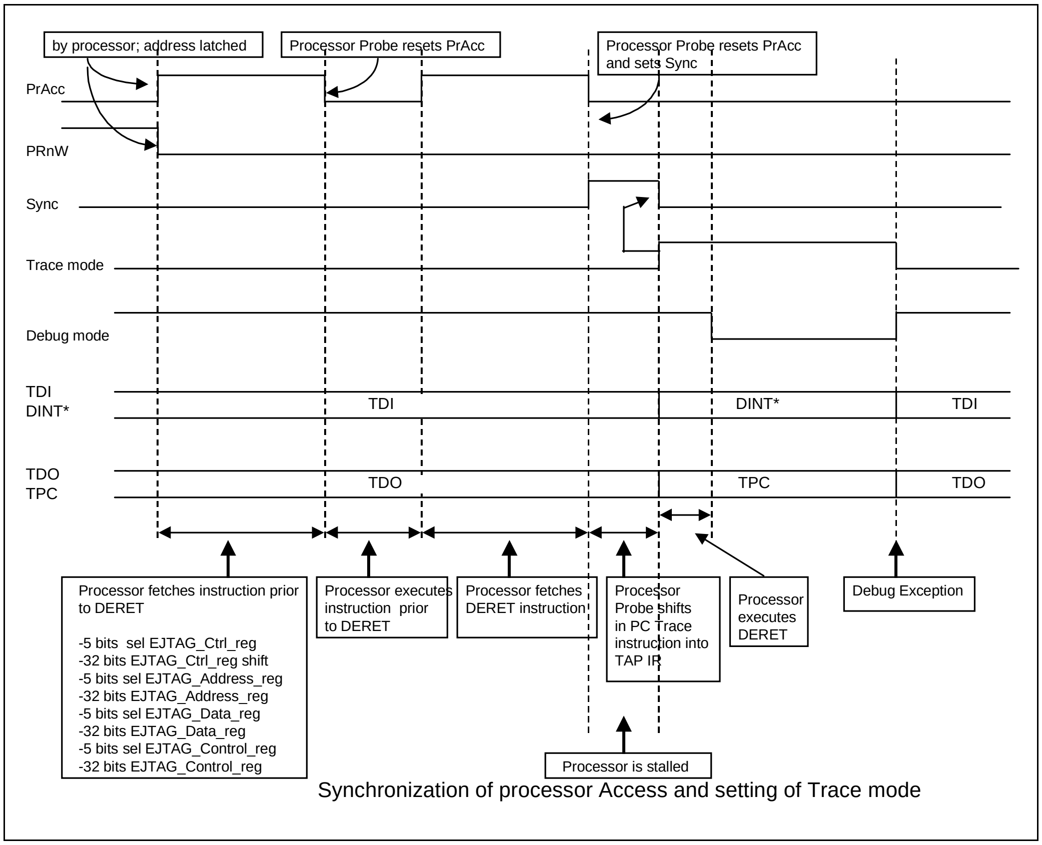

by processor; address latched
Processor Probe resets PrAcc
Processor Probe resets PrAcc and sets Sync
PrAcc
PRnW
Sync
Trace mode
Debug mode
TDI
DINT*
TDI
DINT*
TDI
TDO
TPC
TDO
TPC
TDO
Processor fetches instruction prior to DERET
-5 bits sel EJTAG_Ctrl_reg
-32 bits EJTAG_Ctrl_reg shift
-5 bits sel EJTAG_Address_reg
-32 bits EJTAG_Address_reg
-5 bits sel EJTAG_Data_reg
-32 bits EJTAG_Data_reg
-5 bits sel EJTAG_Control_reg
-32 bits EJTAG_Control_reg
Processor executes instruction prior to DERET
Processor fetches DERET instruction
Processor Probe shifts in PC Trace instruction into TAP IR
Processor executes DERET
Debug Exception
Processor is stalled
Synchronization of processor Access and setting of Trace mode

Figure G.8 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 TDO/TPC pin provides non-sequential program counter output at the DCLK speed. The 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.

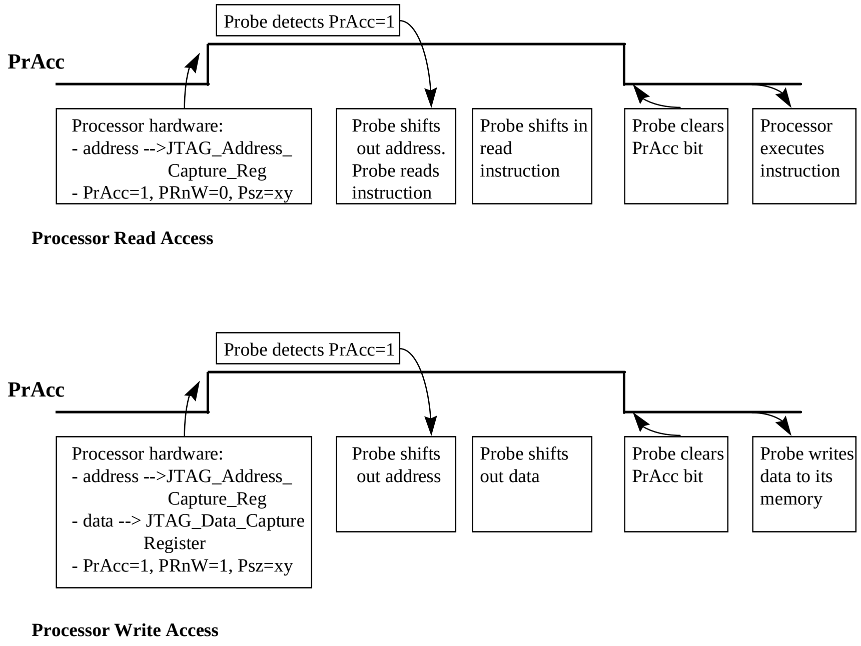
Processor Write Access

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 G.9 depicts the processor and probe actions for the Processor Read and Processor Write Access.

.


Probe detects PrAcc=1
PrAcc
Processor hardware:
- address -->JTAG_Address_Capture_Reg
- PrAcc=1, PRnW=0, Psz=xy
Probe shifts out address. Probe reads instruction
Probe shifts in read instruction
Probe clears PrAcc bit
Processor executes instruction

Processor Read Access

Probe detects PrAcc=1
PrAcc
Processor hardware:
- address -->JTAG_Address_Capture_Reg
- data --> JTAG_Data_Capture Register
- PrAcc=1, PRnW=1, Psz=xy
Probe shifts out address
Probe shifts out data
Probe clears PrAcc bit
Probe writes data to its memory

Processor Write Access

Figure G.9 EJTAG Processor Access

Reset Overview

The processor core, processor peripherals, EJTAG module and the DSU can be reset as follows (see also Figure G.10):

- *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

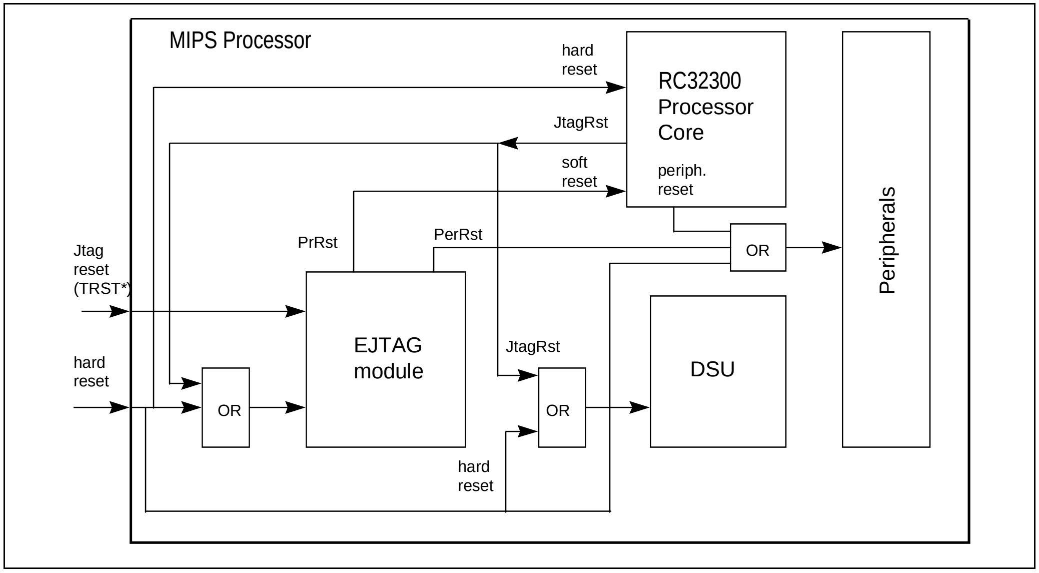


Figure G.10 Reset Overview

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 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 G.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 G.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 G.11	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 G.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.

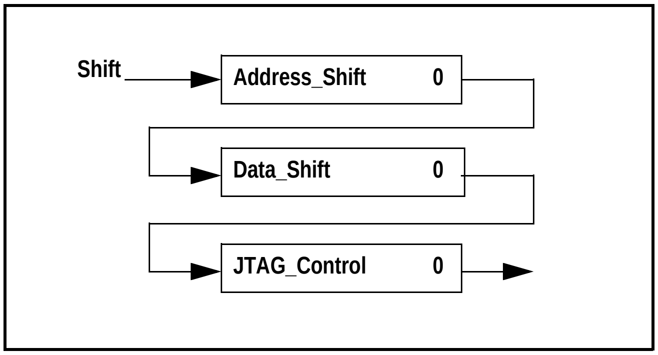


Figure G.11 Shift Order Sequence of the JTAG_All_IR Register

The Debug Unit

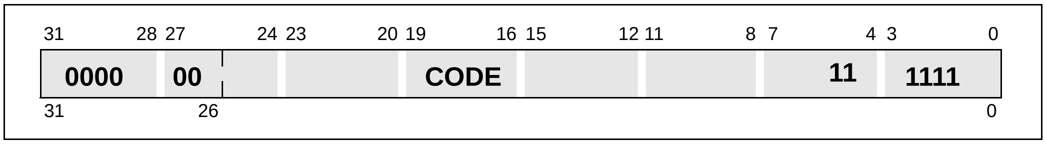
The Debug Unit section describes the debug unit implemented in the processor, and covers the extended instruction to MIPS ISA instruction set as well as support functions and registers for Real Time Debugging.

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 RC32364 implements the following opcode

Notes

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
0000	00		CODE			00	1110
31	26						0

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)

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
0100	00 1 0	0000	0000	0000	0000	0001	1111
31	26 25 24						0

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 RC32364 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 RC32364, refer to the Exception Vector Locations section in Chapter 5 (Table 5.16) of this manual.**	R	0

Table G.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 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 G.6 Debug Register (Part 2 of 2)

Debug Exception Program Counter Register (DEPC)

For the RC32364, 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 RC32364.	R/W	Undefined

Table G.7 Debug Exception Program Counter

Notes

Debug Exception Save Register (DESAVE)

In the RC32364, 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 G.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 G.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 G.9 32-bit Register Map (Base Address = 0xff30 0000)

Notes

Debug Control Register

The Debug Control Register is located at address-offset 0x0000.

Bit(s)	Mnemonic	Description	R/W	Reset
0	TM	Trace Mode 0: This mode will output PC trace information at the TDO/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 TDO/TPC pin. The real-time behavior of the processor is not guaranteed. The RC32364 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 Interrup**t (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 RC32364 executes the WAIT instruction, this bit is set.	R	0
31		**Unused**	R	0

Table G.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 ... 1111: Channel 15	R	0010
31..28	reserved	reserved	R	0

Table G.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 G.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 G.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 G.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 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 PCST(2:0) pins.	R/W	0
31..3	reserved	reserved	R	0

Table G.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 .... 1111: Channel 15	R	0001
31..28	reserved	reserved	R	0

Table G.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 G.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 .... 1111: Channel 15	R	0001
31..28	reserved	reserved	R	0

Table G.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 G.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 G.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 G.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 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 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 G.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 G.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 G.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 TDO/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 (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 RC32364, the number of TPC bits output is 30. To reduce the information at the 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 TDO/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 TPC pin(s), in conjunction with trace information at the 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 TPC pin(s).*

PC Status and Exception Vector Encoding

PC Status Encoding

The PC Trace Status (PCST) Information is output at the same rate as the CPU pipeline clock. The PC status is only active in Real-Time mode. The 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 TPC pin(s).

Table G.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 G.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 like ly 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 PCST shows the EXP code, the 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 TPC pin(s).

Exception Vector Encoding for RC32364:

Table G.24 shows the 4 bit exception code output at the TPC pin(s) during the EXP code at the 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 G.24 Exception and Exception Codes at 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 G.24 Exception and Exception Codes at TPC (Part 2 of 2)

External Interface Definition

EJTAG

The following signals are used during the PC Trace mode (Table G.25 shows the complete list of EJTAG interface signals).

- ***TDO/TPC***: *During PC Trace Mode, the TDO/TPC provides a non-sequential PC (TPC) at the processor clock. TPC is output simultaneously with the Program Counter Trace Status Signals PCST(2:0), starting with PC address 2 or 1. depending on the support of MIPS16 or not.*
- ***PCST(2:0):*** *PC Status Trace Information, with the encoding described in Table G.23.*
- ***DCLK:*** *Processor Clock: This signal is used by the external EJTAG Probe to capture the TPC and PCST signals at the MIPS CPU clock rate. The TPC and PCST signals are output at the positive edge of DCLK.*

Priority of Target Address Output (TPC)

The target address output at TDO/TPC may change due to occurrence of an exception or of a next Jump or Branch instruction. There are priorities specified at which the 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 TPC Output (TM=‘0’ in DCR[0])

During real-time 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 TPC being output, the target address of a taken jump will be output at TDO/TPC, also when it is a Direct Jump. The PCST pins will show the JMP code (see Figure G.12).
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 G.13).
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 G.15).
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 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 G.12). 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, TPC outputs exception code first and then the target address.

Notes

Non-Real Time 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 TPC, this is however enough information to completely reconstruct the program execution flow. The RC32364 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 TPC being output, the target address of a taken jump will be output at TDO/TPC, also when it is a Direct Jump. The 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 RC32364 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 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 G.12).
6. If a jump occurs after exception, TPC outputs exception code first and then the target address.

Examples of PC Trace Output

Conditional PC Relative Jump Instruction

Figure G.12 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 TPC output is not in use, the target PC of the beq starts to output and the 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 PCST lines show 'BRT' but there is no TPC output from its target address since it is a Direct Jump and the TPC line is already outputting.

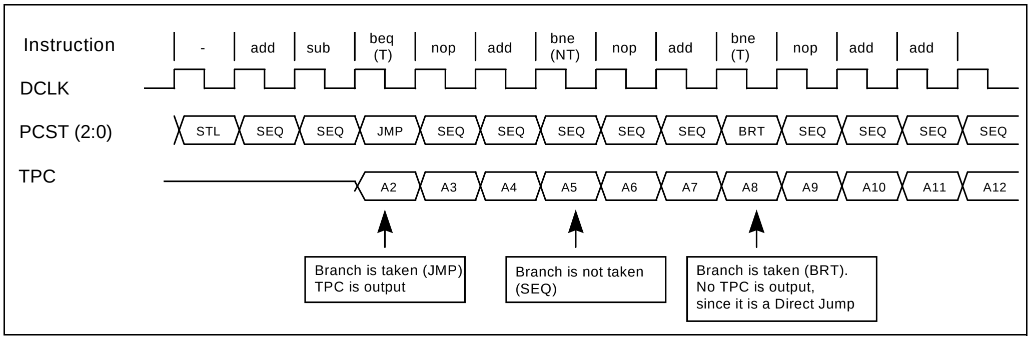


Figure G.12 Trace of Conditional PC Relative Jump Instruction

Indirect Jump Instruction

The execution of an indirect instruction is shown in Figure G.13. When the first indirect jump (jr1) instruction is executed, the processor outputs the 'JMP' code at the PCST pins and starts to output its target address from the lower bit at the 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

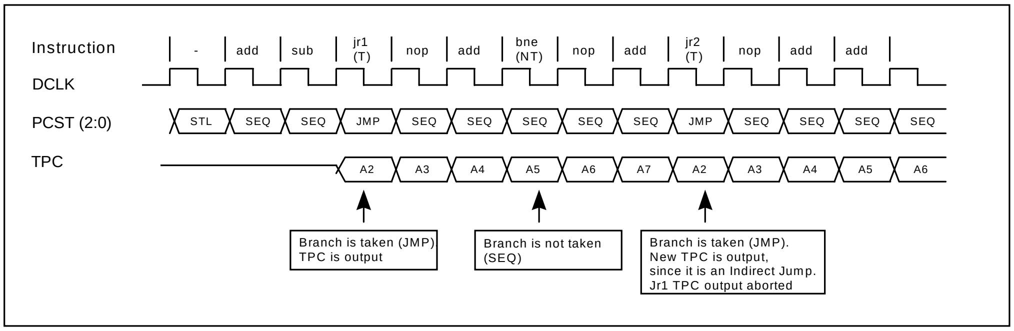

Instruction
- add sub jr1 (T) nop add bne (NT) nop add jr2 (T) nop add add
DCLK
PCST (2:0)
STL SEQ SEQ JMP SEQ SEQ SEQ SEQ SEQ JMP SEQ SEQ SEQ SEQ
TPC
A2 A3 A4 A5 A6 A7 A2 A3 A4 A5 A6
Branch is taken (JMP) TPC is output
Branch is not taken (SEQ)
Branch is taken (JMP). New TPC is output, since it is an Indirect Jump. Jr1 TPC output aborted

Figure G.13 Trace of Indirect Jump Instruction

PC Trace Of An Exception Followed By A Jump Indirect Instruction

In Figure G.14, the Break instruction is executed and causes an exception. This is indicated by the ‘EXP’ code at PCST and the TPC starts outputting the 3-bit exception code ‘001’ starting with the LSB. The taken Jr2 instruction causes the JMP code at PCST and the outputting of its target address at TPC.

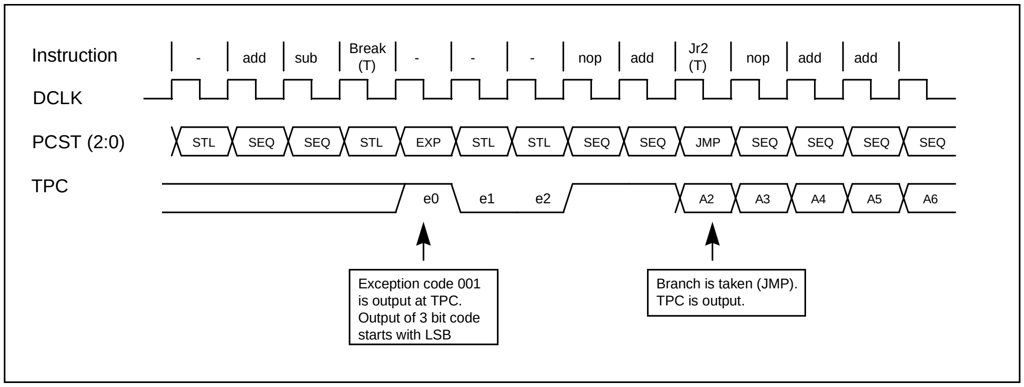

Instruction
- add sub Break (T) - - - nop add Jr2 (T) nop add add
DCLK
PCST (2:0)
STL SEQ SEQ STL EXP STL STL SEQ SEQ JMP SEQ SEQ SEQ SEQ
TPC
e0 e1 e2 A2 A3 A4 A5 A6
Exception code 001 is output at TPC. Output of 3 bit code starts with LSB
Branch is taken (JMP). TPC is output.

Figure G.14 Trace of an Exception Followed by a Jump Indirect Instruction

PC Trace of an Indirect Instruction Followed by an Exception

In Figure G.15, the indirect jump Jr1 starts the 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.

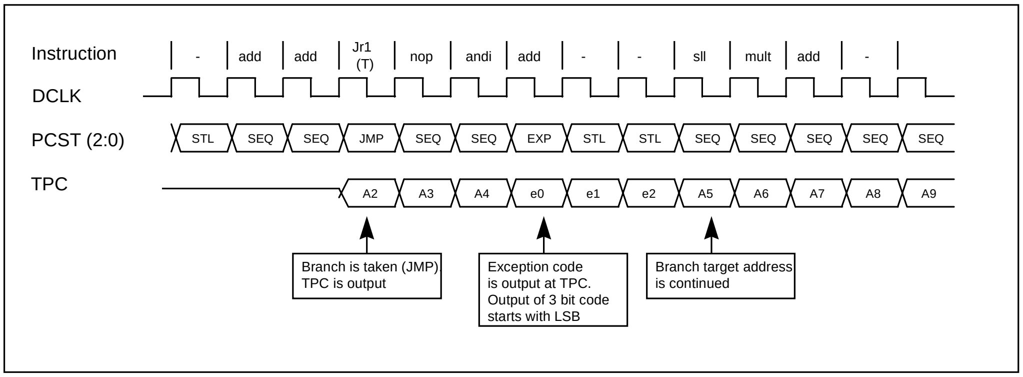

Instruction
- add add Jr1 (T) nop andi add - - sll mult add -
DCLK
PCST (2:0)
STL SEQ SEQ JMP SEQ SEQ EXP STL STL SEQ SEQ SEQ SEQ SEQ
TPC
A2 A3 A4 e0 e1 e2 A5 A6 A7 A8 A9
Branch is taken (JMP) TPC is output
Exception code is output at TPC. Output of 3 bit code starts with LSB
Branch target address is continued

Figure G.15 Trace of Indirect Jump Instruction Followed by an Exception

Notes

Examples of Trace Trigger Output

Trace trigger information is output at the 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 G.16 shows the occurrence of the Trace Trigger TSQ code at the PCST pins for the instruction address that matches the required conditions.

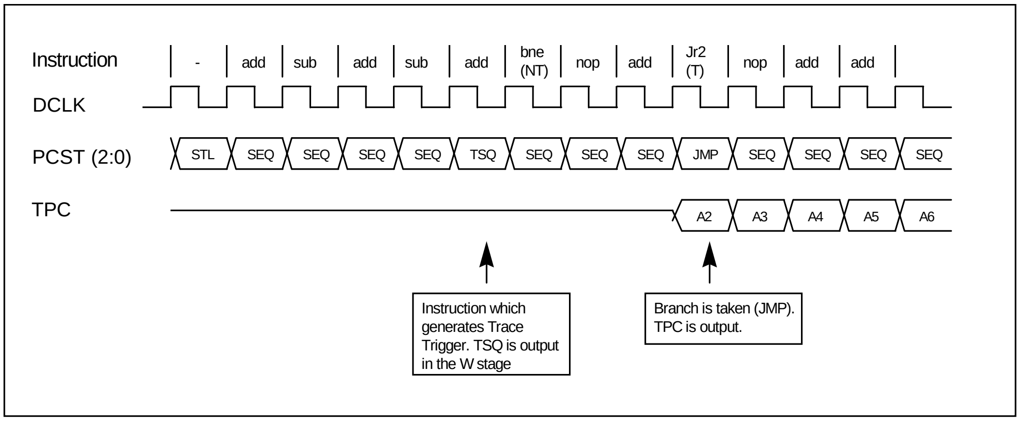


Figure G.16 instruction Address Trace Trigger

Trace Trigger and General Exception at the Same Time

In Figure G.17, both the Trace Trigger and an exception occur at the same moment, then the PCST pins show the TST code, followed by the EXP code. The 3 bit exception code is output at TPC.

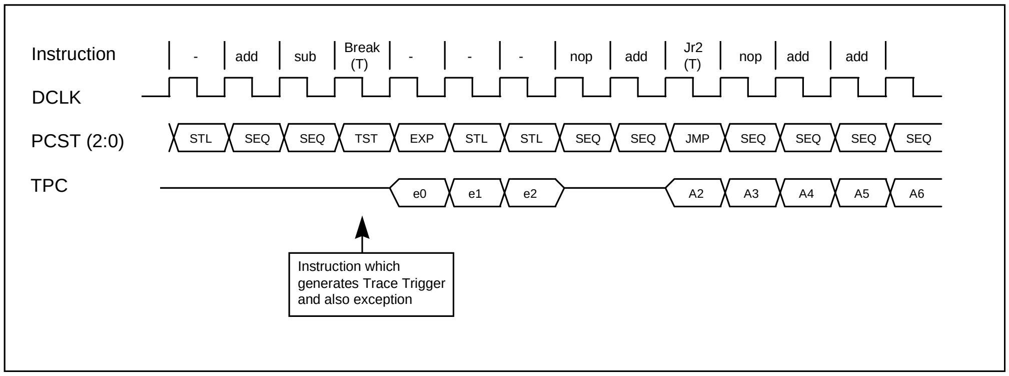


Figure G.17 Trace Trigger and General Exception at the Same Time

Jump Indirect Causes Trace Trigger

In Figure G.18 the Jump Indirect (Jr2) is the instruction that generates the Trace Trigger. This indicated by the TSQ code at the PCST pins.

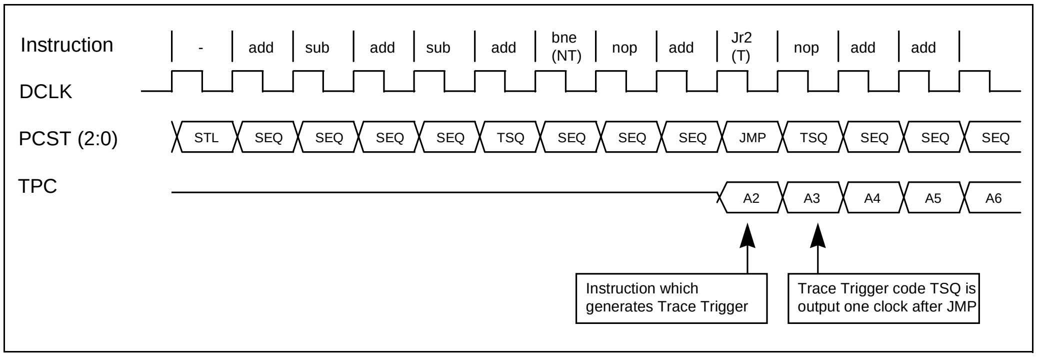


Figure G.18 Jump Indirect Causes Trace Trigger

Instruction after Jump Indirect Causes Trace Trigger

In Figure G.19 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.

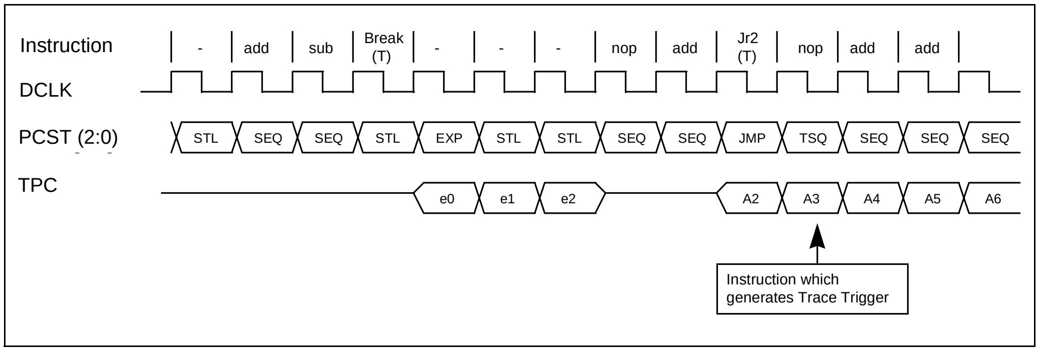


Figure G.19 Instruction after Jump Indirect Causes Trace Trigger

Switching from Real-Time Trace to Debug

Real-Time Trace Mode to Debug Mode (No TPC Output)

In Figure G.20, 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 PCST just before debug mode is entered.

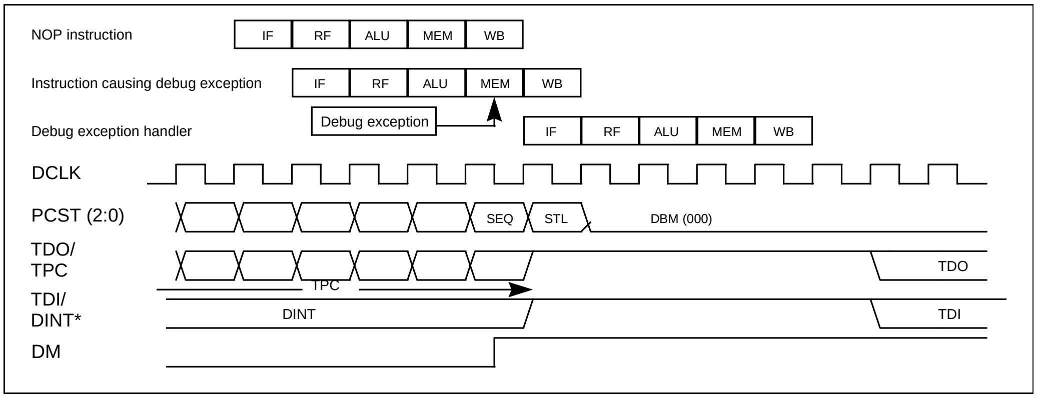


Figure G.20 Real-Time Trace Mode to Debug Mode (No Tpc Output)

Notes

Real-Time Trace Mode to Debug Mode

In Figure G.21, 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 PCST; the debug mode entry is indicated by the DBM code. In debug mode, the TDO/TPC pin function changes from TPC to TDO; TDI/DINT* pin function changes from DINT* to TDI.

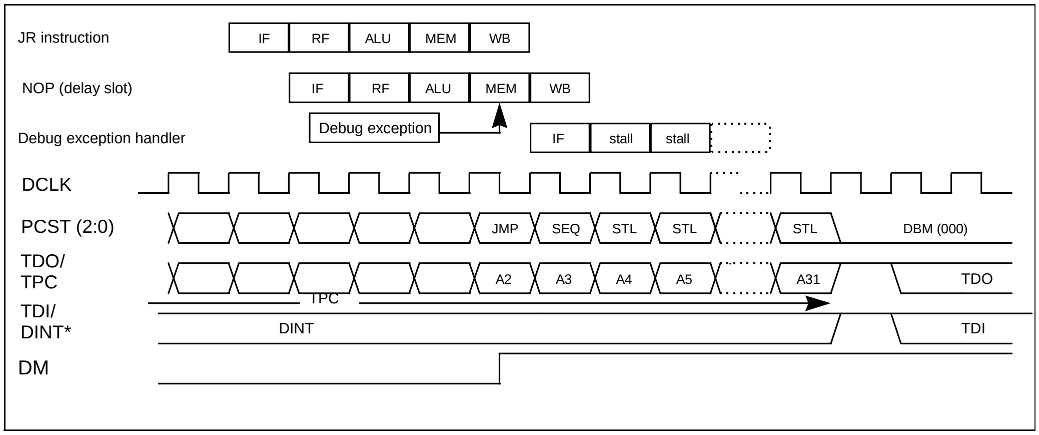
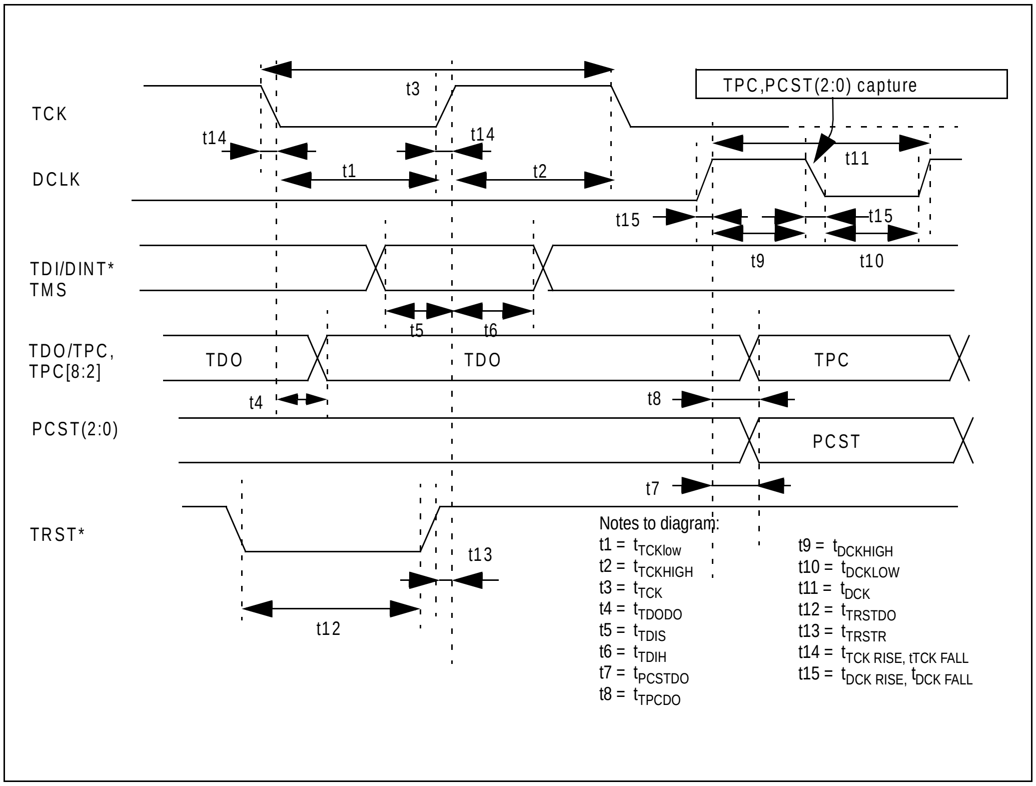


Figure G.21 Real Time Trace Mode to Debug Mode (Debug Exception in Branch Delay Slot)

Pin Out of the Standard EJTAG

Figure G.22 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.



Notes to diagram:

t1 = t_{TCKlow}
t2 = $t_{TCKHIGH}$
t3 = t_{TCK}
t4 = t_{TDODO}
t5 = t_{TDIS}
t6 = t_{TDIH}
t7 = t_{PCSTDO}
t8 = t_{TPCDO}

t9 = $t_{DCKHIGH}$
t10 = t_{DCKLOW}
t11 = t_{DCK}
t12 = t_{TRSTDO}
t13 = t_{TRSTR}
t14 = $t_{TCK\ RISE,\ tTCK\ FALL}$
t15 = $t_{DCK\ RISE,}$ $t_{DCK\ FALL}$

Figure G.22 Timing Diagram of the EJTAG Interface Signals

Notes

Table G.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[a]
1	TRST* (optional)	Input	10 kΩ pull-up resistor
3	TDI, DINT*	Input	10 kΩ pull-up resistor
5	TDO/TPC	Output	33 Ω series resistor
7	TMS	Input	10 kΩ pull-up resistor
9	TCK	Input	10 kΩ pull-up resistor[b]
11	TRST*	Input	10 kΩ pull-up resistor
13	PCST[0]	Output	33 Ω series resistor
15	PCST[1]	Output	33 Ω series resistor
17	PCST[2]	Output	33 Ω series resistor
19	DCLK	Output	33 Ω series resistor
21	DebugBoot	Input	10 kΩ pull-down resistor
23	VIO	Input	Must be connected to the VCC I/O supply of the device.

Table G.25 Pin Numbering of the JTAG and EJTAG Target Connector

[a] The value of the series resistor may depend on the actual PCB layout situation.

[b] 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 G.23 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.

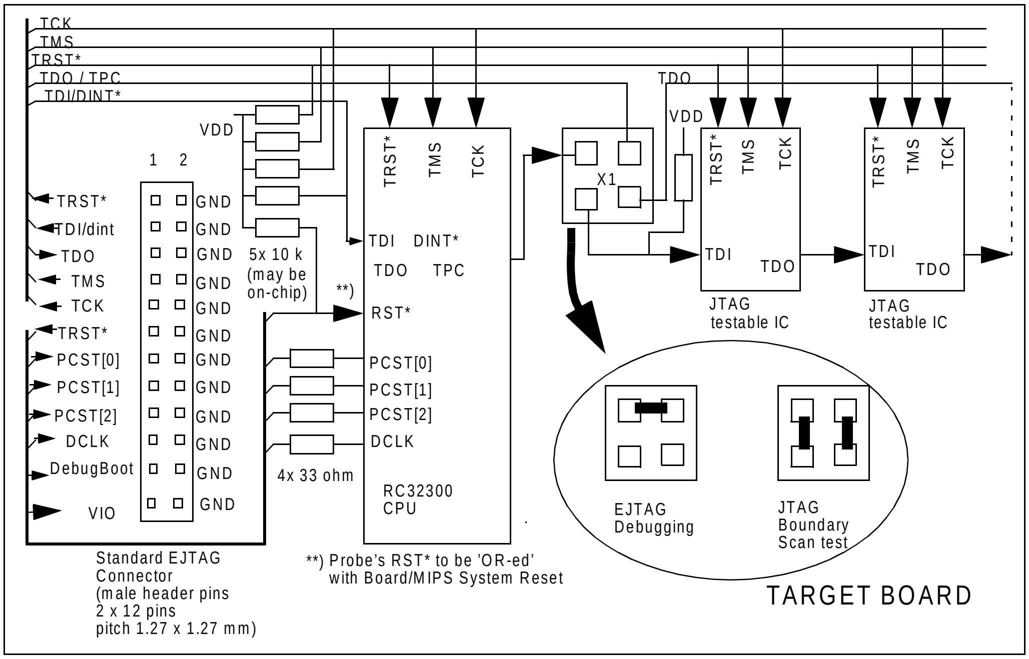


Figure G.23 Application Diagram of Target Board and EJTAG Connection

Notes

Jumper block X1 on the Target Board provides the connection of the processor's TDO/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) 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: TDO/TPC outputs its serial data to the next following TDI input and the 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 (TPC, PCST(2:0), 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 TRST, TDI/DINT*, TMS and 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.

Index