MiniRISC® CW4011 Superscalar Microprocessor Core Technical Manual

A CoreWare® Product

LSI LOGIC®

Order Number C14040.A

Document DB14-000064-01, Second Edition (May 1999)
This document describes revision A of LSI Logic Corporation's MiniRISC® CW4011 Superscalar Microprocessor Core and will remain the official reference source for all revisions/releases of this product until rescinded by an update.

To receive product literature, call us at 1.800.574.4286 (U.S. and Canada); +32.11.300.531 (Europe); 408.433.7700 (outside U.S., Canada, and Europe) and ask for Department JDS; or visit us at http://www.lsilogic.com.

Contents

Figures

Tables

Preface

This book is the primary reference and technical manual for the MiniRISC CW4011 Superscalar Microprocessor Core, referred to in this document as the CW4011 or as the core. This book contains a complete functional description of the CW4011.

Audience

This book is intended for use by engineers and managers who are evaluating the CW4011 core, or for engineers who are designing with this core. This book assumes that this audience is familiar with the concepts of microprocessors and related support devices.

Organization

This book has the following chapters and a glossary of terms.

- Chapter 1, **Introduction**, provides an overview of the CW4011 core and describes the features of the LSI Logic CoreWare® program.
- Chapter 2, **Architectural Overview**, describes the CPU pipeline and microarchitecture, the instructions set architecture, the system coprocessor (CP0), memory management, exception processing, and cache maintenance.
- Chapter 3, **Instruction Set**, describes the MIPS R-series instructions and the instruction set extensions supported in the CW4011 core.
- Chapter 4, **CW4011 Exception Processing**, describes how the CW4011 handles exception processing.
- Chapter 5, **CW4011 Memory Management**, provides detailed information about CP0 and the CW4011 memory management system.

- Chapter 6, **CW4011 Caches**, provides detailed information about the CW4011 caches and cache maintenance.
- Chapter 7, **CW4011 Signals**, describes the CW4011 core I/O signals.
- Chapter 8, **Interface Operation**, describes the main timing scenarios for CW4011 transactions.
- Chapter 9, **ICEport**, describes the CW4011 SerialICE port building block.
- Chapter 10, **Specifications**, contains physical specifications and AC timing for the CW4011 core.
- Appendix A, **CW4011 Register Summary**, provides an overview of all core registers and general MIPS register architecture.
- Appendix B, **Cache Sizing and Design Concerns**, provides information about connecting and selecting different CW4011 cache sizes.
- Appendix C, **Programmer's Notes**, provides information that is useful if you are writing software for the CW4011 core.

Related Publications

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

LR4500 Superscalar Microprocessor Technical Manual, LSI Logic Corporation, Document No. DB14-000068-00.

BDMR4011 Evaluation Board User's Guide, LSI Logic Corporation, Document No. DB15-000055-00.

MIPS SerialICE™ User's Guide, available from LSI Logic Corporation.

Conventions Used in This Manual

The term "word" is used to define a 32-bit quantity, either signed or unsigned. This means that in the CW4011 core a word consists of four 8-bit bytes; a doubleword has 64 bits, or eight 8-bit bytes; and a halfword has 16 bits, or two 8-bit bytes.

Hexadecimal numbers are indicated by the prefix 0x before the number, for example, 0x32CF. Binary numbers are indicated by the prefix 0b before the number, for example, 0b0111 0011 0000.

The following signal conventions are used throughout the manual:

- Active-LOW signals have a lowercase “n” at the end of the signal name (for example, RESETn). Active-HIGH signals have a lowercase “p” at the end of the signal name (for example, SCAop).
- The term “assert” means to drive a signal true or active. The term “deassert” means to drive a signal false or inactive.

Revision History

This table details the changes in this manual over the document’s history. It is not intended to reflect all changes, but should be used as a revision overview.

Document Version	Release Date	Comments
Preliminary	July 1997	Initial release. This document was generated from LSI Logic’s *MiniRISC® CW4010 Superscalar Microprocessor Core Technical Manual*.
Final	October 1997	Final release for revision A. Added Chapter 10, “Specifications,” and revised Appendix B, “Cache Sizing and Design Concerns,” to reflect the proper CW4011 cache connections. Minor modifications made to other chapters.

Chapter 1
Introduction

This chapter introduces the LSI Logic CoreWare program and describes its features. It also provides an overview of the CW4011 core. This chapter contains the following sections:

1.1 CW4011 Overview

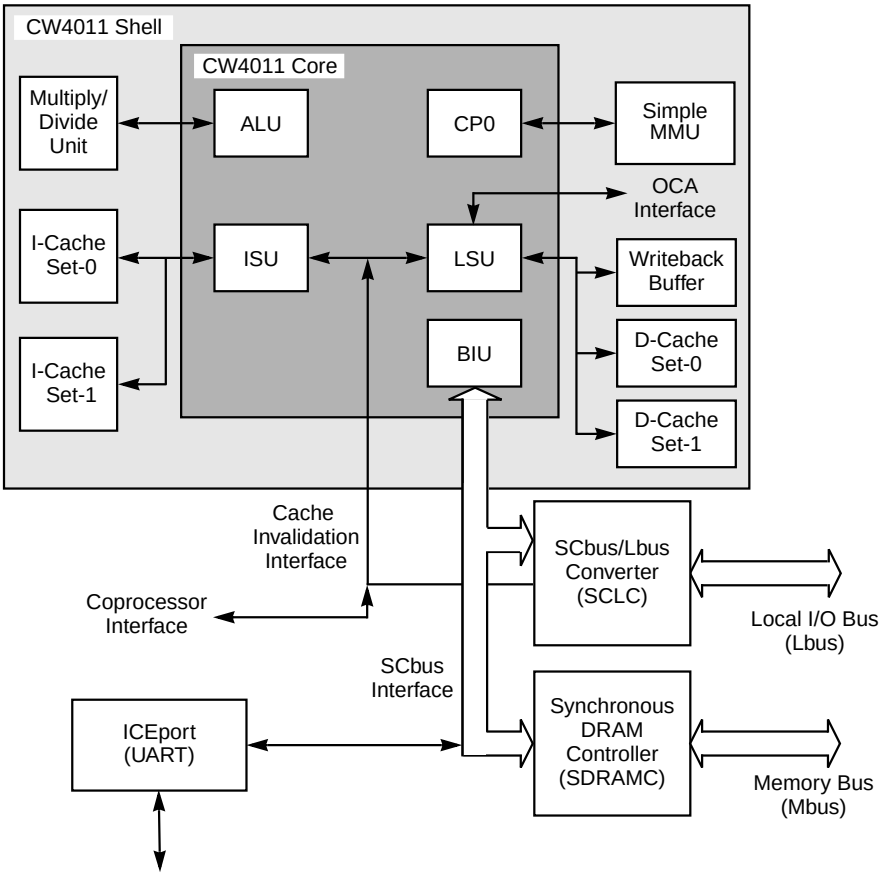
LSI Logic Corporation has developed the MIPS II–compatible MiniRISC CW4011 Superscalar Core using LSI Logic’s CoreWare system-on-a-chip methodology. The CW4011 is a member of LSI Logic’s MiniRISC family, the next generation of MIPS RISC products.

You can use the superscalar CW4011 as a microprocessor core in products that require higher performance than that of an LSI Logic CW400x microprocessor core. The CW4011 core is available as a CoreWare product for use in customer ASIC designs, and is also used in LSI Logic’s ASSPs (application specific standard products), such as the ATMizer® II ATM-SAR chip.

1.2 CW4011 Core and Building Blocks

As shown in Figure 1.1, the CW4011 is implemented at two levels: the standard CW4011 core and the optional shell.

Figure 1.1 CW4011 Core and Building Blocks Diagram



1.2.1 CW4011 Core

The CW4011 superscalar microprocessor core is an encrypted synthesizable Verilog (or VHDL) model. It is process independent and made up of the following units:

- Arithmetic logic unit (ALU)
- System control coprocessor (CP0)
- Bus interface unit (BIU)
- Load store unit (LSU)
- Instruction scheduler unit (ISU)

1.2.2 CW4011 Shell

The following microprocessor building blocks are available with the basic microprocessor core and are available as part of the CW4011 shell. The shell is an unencrypted Verilog model that can include:

- Direct-mapped or two-way set associative instruction cache (I-cache) with cache sizes selectable up to 16 Kbytes
- Direct-mapped or two-way set associative data cache (D-cache) with cache sizes selectable up to 16 Kbytes
- Simple memory management unit (MMU) with an optional translation lookaside buffer (TLB)
- Standard multiply/divide unit (MDU) or a high-performance multiply/accumulate unit (MAC)
- WriteBack buffer for writeback cache mode

1.2.3 Other CW4011 Components

The following components are typically included for any CW4011 design and are implemented in the LR4500 reference device.

- ICEport (UART) for SerialICE™ hardware and software debugging support. A SerialICE manual is available from LSI Logic upon request.
- SCBus/Lbus (SCLC) converter for off-chip components.

- Synchronous DRAM controller (SDRAMC) to interface the core SCbus with off-chip memory.

Please note the following two considerations for any CW4011 microprocessor core design:

- *LSI Logic provides the SDRAMC and SCLC modules as source code only. LSI Logic does not supply product support or documentation for these optional building block modules.*
- *The MMU and MDU components are considered part of the building blocks shell and are not open to users. To modify either the MDU or MMU for your design, please contact LSI Logic.*

1.2.4 Interfaces

The core has four major interfaces:

- The coprocessor interface connects the core with up to three coprocessors (CP1, CP2, and CP3), as well as the internal coprocessor (CP0).
- The cache invalidation interface connects the core with optional cache coherency logic. The core uses this bus to communicate only with the on-chip caches.
- The SCbus, the bidirectional system bus, allows the CW4011 to communicate with system elements outside the core, such as the SCLC and the SDRAMC.
- The OCAbus interface allows on-chip access (OCA) to on-chip modules at the Cache Read pipeline stage without going through the SCbus.

1.2.5 Related Modules

In addition to the core, the MiniRISC product family includes a variety of other modules including:

- LSI Logic's MiniSIM® performance simulator
- Verilog and VHDL models
- A system verification environment (SVE)
- A PROM monitor

- Third party software support (Compiler and RTOS)
- LR4500 evaluation chip
- Evaluation boards for concurrent software development
- LSI Logic's CoreWare, described in Section 1.4, "CoreWare Program"

1.3 Features

The CW4011 core has the following features:

- Full MIPS II instruction set implementation (R4000 32-bit mode compatible)
- Instruction set extensions to support embedded applications
- Superscalar execution with up to two instructions issued per clock cycle
- 64-bit on-chip data bus system interface
- High-performance coprocessor interface for user definable coprocessors and high performance hardware floating-point unit (FPU)
- 3.3-Volt operation
- 90-MHz worst-case commercial maximum clock rate using standard cell ASIC
- 130+ Dhrystone MIPS at 90 MHz
- 180 native MIPS peak, 120 native MIPS sustained with standard compiled MIPS code at 90 MHz
- 7.0 mW/MHz core power with power management
- Integrated cache controllers with separate instruction and data caches
  - D-cache set sizes selectable from 1 to 8 Kbytes (up to two sets available)
  - I-cache set sizes selectable from 1 to 8 Kbytes (up to two sets available)
- Optional, modifiable building blocks, such as a MAC and an MMU

- SerialICE scan chain allows full testing in embedded ASIC designs
- Models available:
  - Performance and software development model
  - Verilog and VHDL models (referred to in this manual as HDL models)
  - Gate-level, timing-accurate model in various third party simulation environments
- Compatible with the full range of MIPS and third party software development tools
- Compact basic microprocessor core size—2.5 by 3.5 mm including BIU, cache controllers, and external write buffer
- R3000 compatibility mode for exception handling and status registers

1.4 CoreWare Program

An LSI Logic core is a fully defined, optimized, and reusable block of logic. It supports industry-standard functions and has predefined timing and layout. The core is also an encrypted RTL simulation model for a wide range of VHDL and Verilog simulators.

The CoreWare library contains an extensive set of complex cores for the communications, consumer, and computer markets. The library consists of high-speed interconnect functions such as the GigaBlaze™ G10™ core, MIPS embedded microprocessors, MPEG-2 decoders, a PCI core, and many more.

The library also includes megafunctions or building blocks, which provide useful functions for developing a system on a chip. Through the CoreWare program, you can create a system on a chip uniquely suited to your applications.

Each core has an associated set of deliverables, including:

- RTL simulation models for both Verilog and VHDL environments
- An SVE for RTL-based simulation
- Netlists for full timing simulation
- Complete documentation

- LSI Logic ToolKit support

LSI Logic's ToolKit provides seamless connectivity between products from leading electronic design automation (EDA) vendors and LSI Logic's manufacturing environment. Standard interfaces for formats and languages such as VHDL, Verilog, Waveform Generation Language (WGL), Physical Design Exchange Format (PDEF), and Standard Delay Format (SDF) allow a wide range of tools to interoperate within the LSI ToolKit environment. In addition to design capabilities, full scan automatic test pattern generation (ATPG) tools and LSI Logic's specialized test solutions can be combined to provide high-fault coverage test programs that assure a fully functional design.

Because your design requirements are unique, LSI Logic is flexible in working with you to develop your system-on-a-chip CoreWare design. Three different work relationships are available:

- You provide LSI Logic with a detailed specification and LSI Logic performs all design work.
- You design some functions while LSI Logic provides you with the cores and megafunctions, and LSI Logic completes the integration.
- You perform the entire design and integration, and LSI Logic provides the core and associated deliverables.

Whatever the work relationship, LSI Logic's advanced CoreWare methodology and ASIC process technologies consistently produce Right-First-Time™ silicon.

Chapter 2
Architectural Overview

This chapter discusses the CPU pipeline, CPU architecture, instruction set architecture, the system coprocessor (CP0), memory management, exception processing, and cache maintenance. This chapter is divided into the following sections:

2.1 Architectural Overview

The CW4011 is fully compatible with the R3000 and R4000 32-bit instruction sets (MIPS I and MIPS II), but it also uses an updated hardware architecture to provide higher absolute performance than any other available MIPS core. The CW4011 also provides substantially better instructions-per-clock performance than other MIPS processors. At the same time, the hardware design remains compact in comparison with similar superscalar architectures.

The CW4011 implements a 32-bit virtual address space, with up to 2 Gbytes of virtual address space available to each user-level process. Individual memory locations are byte-addressed.

The CW4011 implements a 32-bit physical address space. Individual memory locations are byte-addressed and, combined with the virtual address space, provide a total of four Gbytes of physical address memory.

The CW4011 can issue and complete two instructions per cycle using a combination of five independent execution units:

- Arithmetic logic unit (ALU)
- Load/store unit (LSU)

  LSU executes load and store instructions. It also executes add and load immediate instructions, allowing an add instruction to be issued with another add or logical instruction.
- Branch unit
- Multiply/shift unit
- Coprocessor interface

  Coprocessor interface can feed an instruction to a customer-defined coprocessor unit. Contact LSI Logic for further information if your design requires a coprocessor.

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

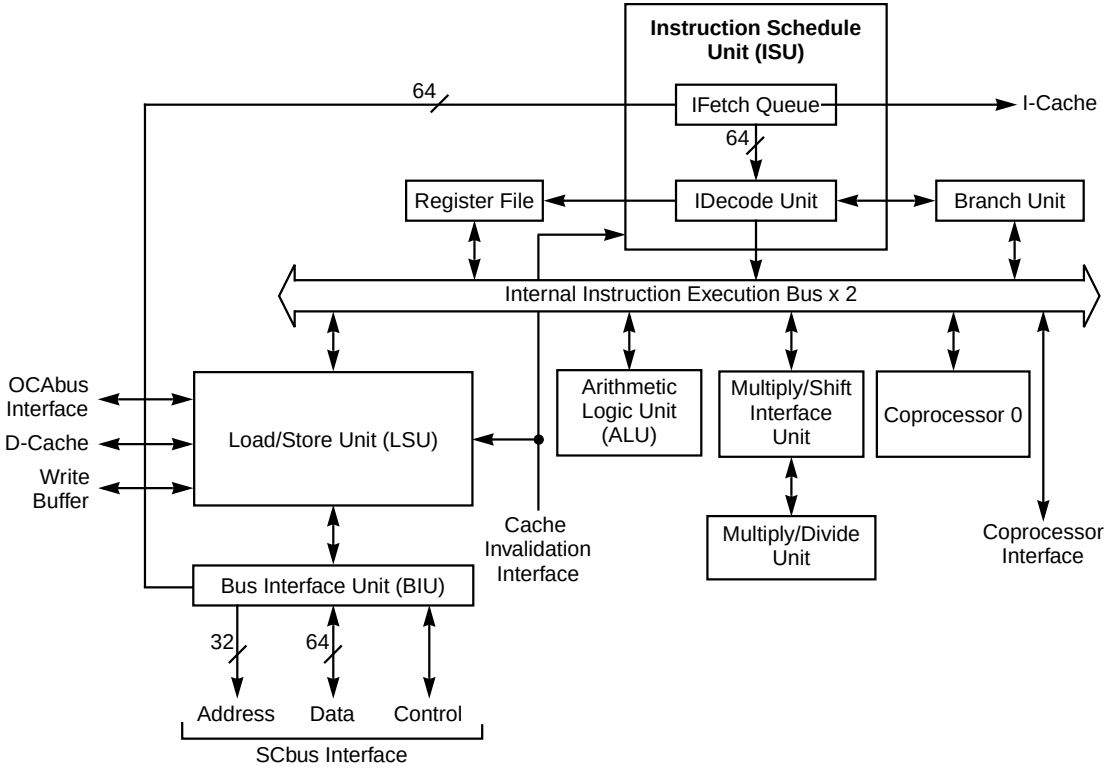
Load instructions have a single hardware delay slot for loads that hit in the cache, but the hardware activates an interlock on register conflicts so that a NOP (no operation) is not required in the delay slot. On a load miss, the CW4011 extends the hardware conflict detection so that if the load data is not required by subsequent instructions in the pipeline, the CPU is not stalled. The operation is called *load scheduling*.

Figure 2.1 shows a block diagram of the basic CW4011 core.

Three units handle instructions:

- The IFetch Queue optimizes the supply of instructions to the microprocessor, even across breaks in the sequential flow of execution (jumps and branches).
- The IDecode Unit decodes the instructions from the IFetch Queue, determines the actions required for the instruction execution, and manages the Register File, LSU, ALU, and Multiply/Divide Units accordingly.
- The Branch Unit is used when branch and jump instructions are recognized within the instruction stream.

Figure 2.1 CW4011 Block Diagram



The Register File contains the core's general purpose registers. (There are 32 general purpose registers located in the CPU. Of these registers 31 are read/write registers and 1 is the zero register.) The Register File supplies source operands to the execution units and handles the storage of results to target registers.

Three units perform logical, arithmetic, and data-movement operations:

- The LSU manages loads and stores of data values. Data values are loaded from either the D-cache or from the SCbus Interface in the event of a D-cache miss. Stores pass to the D-cache and the SCbus interface through the write buffer. The LSU is also able to perform a restricted set of arithmetic operations, including the addition of an immediate offset as required in address calculations.
- The ALU calculates the result of an arithmetic or logical operation.

- The multiply/shift interface unit performs multiply and divide operations. You can select a number of modular options for this unit, including an option with full multiply/accumulate capability.

The CW4011 core has four major interfaces for data transfer:

- The BIU manages the flow of instructions and data between the core and the system by means of the SCbus Interface. This interface provides the main channel for communication between the CW4011 core and the other functional blocks in the system. Some blocks may be implemented as CoreWare library functions integrated on the same die as the microprocessor core; others may be implemented in separate devices connected by means of I/O pins at the board level.
- The coprocessor interface allows tightly coupled special-purpose processing units to be attached to the core, enhancing the microprocessor's general-purpose computational power. Contact LSI Logic for further information if you need a coprocessor in your design.
- The cache invalidation interface allows supporting hardware outside the core to maintain the coherency of on-board cache contents for systems that include multiple main-bus masters.
- The OCAbus interface allows on-chip modules to be accessed at the Cache Read (CR) stage of the pipeline without going through an SCbus transaction. This improves performance since it reduces traffic on the SCbus and therefore reduces latency.

2.2 Cache and External Interface

I-cache control is performed by the ISU. D-cache control is performed by the LSU.

A write buffer is also implemented within the LSU, so that CPU execution need not stall if a number of stores are performed in quick succession. The write buffer accepts the store addresses and data values, and passes them on to main memory as rapidly as it can accept them. During this time, the CPU proceeds with execution.

The BIU provides the interface to on-chip peripherals. One or more peripherals will typically provide a path to off-chip resources, including

main memory. The BIU supports dynamic bus sizing between 32-bit and 64-bit transactions, see Section 2.6.3, “64-bit vs. 32-bit Memory Interface,” for more information on bus sizing.

The on-chip system interface presented by the BIU is the SCbus. This bus has a 64-bit data bus and a 32-bit address bus. Address and data are not multiplexed. I-cache and D-cache refills use the 64-bit data bus to achieve the highest performance possible.

2.3 Clocking and Power Management

The CPU core is clocked by a single phase, 1x clock with a 40–60% duty cycle requirement. Applications that require a slower system clock interface may use a phase-locked loop circuit (PLL), available as a cell in LSI Logic’s ASIC libraries, and logic to implement a clock multiplier circuit for the CPU.

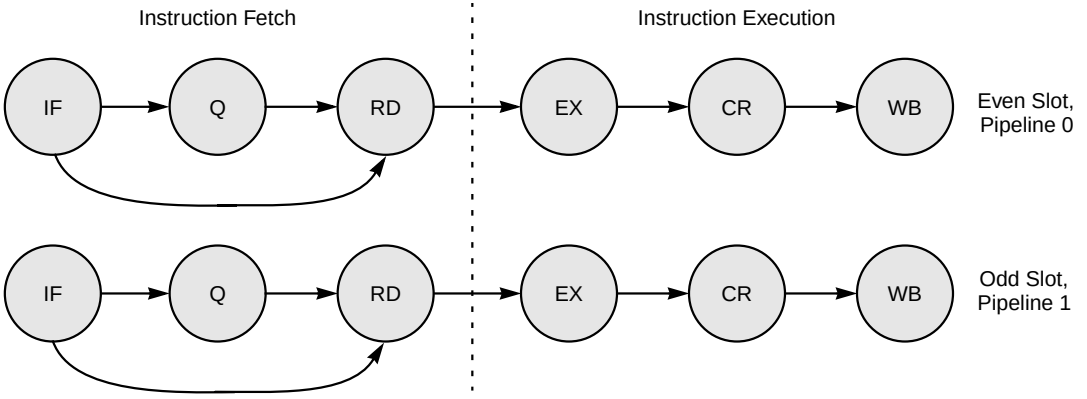
Power management is provided for the CPU by the WAITI (Wait for Interrupt) instruction and by gating the clock separately for each functional unit. Units are clocked only when needed. In addition, the core and cache RAMs are static, so that the clock may be slowed or turned off by user logic to save power.

2.4 Pipeline Architecture

This section describes the CPU pipelines, instruction fetching and scheduling. It also contains an instruction set summary.

As shown in Figure 2.2, the CW4011 core has two identical concurrent five-stage pipelines that provide the core with its superscalar capabilities. One pipeline is known as the even slot or pipeline 0, and the other as the odd slot or pipeline 1.

Figure 2.2 CW4011 Instruction Pipeline



The first two pipeline stages (and conditional Q stage) are used during instruction fetch and the last three stages during instruction execution. Once a stage has accepted an instruction from the previous stage it must hold the instruction for re-execution in case the pipeline stalls. The function of each pipeline stage is summarized below.

IF (Instruction Fetch) – The CW4011 fetches the instruction during the first stage.

Q (Queuing) – Instructions may enter this conditional stage if they deal with execution unit or register conflicts. An instruction that does not cause an execution unit or register conflict is fed directly to the RD stage.

RD (Read) – During this stage, any required operands are read from the Register File while the instruction is decoded.

EX (Execute) – All instructions are executed in this stage. Conditional branches are resolved in this cycle. The address calculation for load and store instructions is performed in this stage.

CR (Cache Read) – This stage is used to access the cache for load and store instructions. Data is returned to the register bypass logic at the end of this stage.

WB (WriteBack) – Results are written into the Register File during this stage.

Sections 2.4.1 and 2.4.2 provide more detailed information about pipeline transactions.

2.4.1 Instruction Fetch and Scheduling

The IF, Q, and RD stages fetch two instructions per cycle and issue them to the EX stage. The CW4011 fetches instructions as doubleword aligned pairs (even and odd). There is a two-instruction window in the RD stage during the instruction decode operation. When only the even slot can be scheduled because the odd slot has a dependency, the window slides down one instruction. In other words, although instructions are always fetched as doubleword pairs, they are scheduled on single-word boundaries.

The primary purpose of the Q stage is to execute branch instructions with minimal penalty. The CW4011 generally fills the Q stage whenever the RD stage has to stall. This occurs fairly frequently on typical compiled code, because of register conflicts, cache misses, and resource conflicts. Filling the Q stage in these cases allows the IF stage to work ahead one cycle.

If a branch instruction is encountered when the Q stage is already active, it is predicted that the branch will be taken. The IF stage does not bring in any more instructions following the current address, but instead begins fetching those instructions starting at the branch target address. At this point, the Q stage still holds the pair of instructions immediately following the pair that contained the branch.

The branch target enters the RD stage, bypassing the Q stage, as shown in Figure 2.2. The branch prediction logic in the ISU resolves the branch condition when the branch instruction enters the EX stage. If the branch prediction logic predicts the branch correctly, the instructions in the Q stage are cancelled. If it predicts the branch incorrectly, the ISU cancels the branch target. In this case, it takes non-branch sequential instructions from the Q stage and restarts the IF stage at the non-branch sequential stream. The process is different when the branch instruction is in the odd instruction slot.

If the branch prediction logic correctly predicts a branch in the even instruction slot when the Q stage is full, there is generally no cycle penalty associated with it. If the branch prediction logic predicts the branch incorrectly, the branch has a one cycle penalty.

If the branch instruction was in the odd instruction slot, the branch delay slot instruction always executes by itself and has no chance to fill the other execution slot. There may be some advantage to a software assembler that can attempt to place branches in even word addresses.

The branch prediction logic must be able to look at two instructions at the same time, from either the Q latches or the RD latches, depending on whether the Q stage is active. When it looks at the two instructions, if one is a branch, it passes the offset in that instruction into a dedicated adder to calculate the branch address for the IF stage of the instruction fetch. Because this is done speculatively, it also saves the non-branch value of the PC (Program Counter) for the possible restart of the sequential instructions from the Q stage.

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

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

2.4.2 Instruction Execution

During instruction execution, a pair of instructions (or a single instruction when there was a previous block) are individually passed to independent execution units. Each execution unit receives its operands from the register bypass logic and an instruction from the instruction scheduler. Each single cycle instruction spends one RUN cycle in an execution unit, with the result then fed to the register/bypass unit for the CR stage. Please note that multiple cycle instructions may spend longer than one cycle in an execution unit.

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

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

2.5 Instruction Set Summary

Table 2.1 summarizes the instruction set for the CW4011. The CW4011 supports 32-bit MIPS II instructions and implements additional CW4011-specific instructions. If the design includes the optional MMU, the CW4011 supports the TLB instructions. All instructions are 32 bits long. Table 2.1 includes only the MIPS II, CW4011-specific, and TLB

instructions. With the exception of RFE, MIPS I instructions are not shown.

Table 2.1 CW4011 Instruction Set Summary

Op	Description	Op	Description
Load/Store Instructions			
LB	Load Byte	SH	Store Halfword
LBU	Load Byte Unsigned	SW	Store Word
LH	Load Halfword	SWL	Store Word Left
LHU	Load Halfword Unsigned	SWR	Store Word Right
LW	Load Word	LL[1]	Load Linked
LWL	Load Word Left	SC[1]	Store Conditional
LWR	Load Word Right	SYNC[1]	Sync
SB	Store Byte		
ALU Immediate Instructions			
ADDI	Add Immediate	ANDI	AND Immediate
ADDIU	Add Immediate Unsigned	ORI	OR Immediate
SLTI	Set on Less Than Immediate	XORI	Exclusive OR Immediate
SLTIU	Set on Less Than Immediate Unsigned	LUI	Load Upper Immediate
Three-Operand, Register-Type Arithmetic Instructions			
ADD	Add	SLTU	Set on Less Than Unsigned
ADDU	Add Unsigned	AND	AND
SUB	Subtract	OR	OR
SUBU	Subtract Unsigned	XOR	Exclusive OR
SLT	Set on Less Than	NOR	NOR
Shift Instructions			

(Sheet 1 of 4)

Table 2.1 CW4011 Instruction Set Summary (Cont.)

Op	Description	Op	Description
SLL	Shift Left Logical	SLLV	Shift Left Logical Variable
SRL	Shift Right Logical	SRLV	Shift Right Logical Variable
SRA	Shift Right Arithmetic	SRAV	Shift Right Arithmetic Variable
Multiply/Divide Instructions			
MULT	Multiply	MFHI	Move From HI
MULTU	Multiply Unsigned	MTHI	Move To HI
DIV	Divide	MFLO	Move From LO
DIVU	Divide Unsigned	MTLO	Move To LO
Computation Instruction Extensions			
ADDCIU[2]	Add Circular Immediate	SELSR2	Select and Shift Right
FFS2	Find First Set Bit	SELSL2	Select and Shift Left
FFC2	Find First Clear Bit	MADD2	Multiply/Add
MIN2	Minimum	MADDU2	Multiply/Add Unsigned
MAX2	Maximum	MSUB2	Multiply/Subtract
		MSUBU2	Multiply/Subtract Unsigned
Jump and Branch Instructions			
J	Jump	BLEZ	Branch on Less than or Equal to Zero
JAL	Jump And Link	BGTZ	Branch on Greater Than Zero
JR	Jump Register	BLTZ	Branch on Less Than Zero
JALR	Jump And Link Register	BGEZ	Branch on Greater than or Equal to Zero
BEQ	Branch on Equal	BLTZAL	Branch on Less Than Zero And Link
BNE	Branch on Not Equal	BGEZAL	Branch on Greater than or Equal to Zero And Link

(Sheet 2 of 4)

Table 2.1 CW4011 Instruction Set Summary (Cont.)

Op	Description	Op	Description
Branch Likely Instructions			
BEQL[1]	Branch on Equal Likely	BGEZL[1]	Branch on Greater than or Equal to Zero Likely
BNEL[1]	Branch on Not Equal Likely	BLTZALL[1]	Branch on Less Than Zero And Link Likely
BLEZL[1]	Branch on Less than or Equal to Zero Likely	BGEZALL[1]	Branch on Greater than or Equal to Zero And Link Likely
BGTZL[1]	Branch on Greater Than Zero Likely	BCzTL[1]	Branch on Coprocessor z True Likely
BLTZL[1]	Branch on Less Than Zero Likely	BCzFL[1]	Branch on Coprocessor z False Likely
Trap Instructions			
TEQ	Trap on Equal	TLT	Trap on Less Than
TEQI	Trap on Equal Immediate	TLTI	Trap on Less Than Immediate
TGE	Trap on Greater than or Equal	TLTU	Trap on Less Than Unsigned
TGEI	Trap on Greater than or Equal Immediate	TLTIU	Trap on Less Than Immediate Unsigned
TGEU	Trap on Greater than or Equal Unsigned	TNE	Trap if Not Equal
TGEIU	Trap on Greater than or Equal Immediate Unsigned	TNEI	Trap if Not Equal Immediate
Special Instructions			
SYSCALL	System Call	BREAK	Breakpoint
Coprocessor Instructions			
LWCz	Load Word to Coprocessor z	CFCz	Move Control From Coprocessor z
SWCz	Store Word from Coprocessor z	COPz	Coprocessor Operation
MTCz	Move To Coprocessor z	BCzT	Branch on Coprocessor z True

(Sheet 3 of 4)

Table 2.1 CW4011 Instruction Set Summary (Cont.)

Op	Description	Op	Description
Coprocessor Instructions (continued)			
MFCz	Move From Coprocessor z	BCzF	Branch on Coprocessor z False
CTCz	Move Control To Coprocessor z		
System Control Coprocessor (CP0) Instructions			
MTC0	Move To CP0	TLBWI[3]	Write Indexed TLB Entry
MFC0	Move From CP0	TLBWR[3]	Write Random TLB Entry
RFE	Restore From Exception (R3000 mode only)	TLBP[3]	Probe TLB for Matching Entry
ERET	Exception Return (R4000 mode only)	WAITI[2]	Wait for Interrupt
TLBR[3]	Read Indexed TLB Entry		
Cache Maintenance Instructions			
FLUSHD[2,4]	Flush D-Cache	FLUSHID[2,4]	Flush I-Cache and D-Cache
FLUSHI[2,4]	Flush I-Cache	WB[2]	Writeback
(Sheet 4 of 4)			

1. MIPS II instruction.
2. CW4011-specific instruction.
3. Valid only with implemented MMU building block.
4. Do not confuse these instructions with the FLUSH instruction in R6000 processors.

In addition to the standard MIPS II instruction set, the CW4011 implements certain instruction set extensions, shown in Table 2.2, that provide greater application code performance for typical embedded applications. Instruction set extensions are included only if they significantly improve performance, have no impact on clock cycle rate, and have minimal impact on the size and complexity of the hardware.

Table 2.2 Instruction Set Extensions

Extension	Format and Description
Find First Set, Find First Clear	`FFS rd,rs, FFC rd,rs` These instructions, respectively, find the first set bit and the first clear bit in the source register, and return the bit number to the destination register. They are useful for many applications such as interrupt handlers, floating point emulation, and graphics.
Select and Rotate Left, Select and Rotate Right	`SELSL rd,rs,rt, SELSR rd,rs,rt` These instructions select 32 bits from the 64-bit source register pair and rotate the selected data left or right by the number of bits specified in the new CP0 Rotate register. They are useful for data alignment operation in graphics and in bit-field selection routines for data transmission and compression applications.
Add Circular Immediate	`ADDCIU rt,rs,immediate` This instruction does an immediate add, modified according to the value in the new CP0 CMask register. It is useful in addressing circular buffers. This instruction is important in DSP (digital signal processing) and other applications that use circular buffers.
Multiply/ADD, Multiply/SUB Instructions	`MADD(U) rs,rt, MSUB(U) rs,rt` These instructions are useful in many signal processing and graphics transform algorithms. Only implemented with the high-performance multiply/accumulate unit, these instructions do a 32 x 32 multiply and then either add or subtract the result to the 64-bit HI/LO register pair.
Wait for Interrupt	`WAITI` This instruction halts the CPU in a power saving mode until one of the hardware interrupt lines becomes active. Upon interrupt, normal execution is resumed starting at the interrupt vector address.
Minimum	`MIN rd, rs, rt` The source operands `rs` and `rt` are compared as two's complement values. The smaller value is stored in the `rd` register.
Maximum	`MAX rd, rs, rt` The source operands `rs` and `rt` are compared as two's complement values. The larger value is stored in the `rd` register.

2.6 Configurability and Options

The CW4011 is implemented using Verilog HDL (Hardware Description Language) as the design source, and the LSI Logic Standard Cell Library and layout tools for physical design. You can easily modify and configure the CW4011 core to meet specific design requirements. The options available in the basic core are shown in the following sections. Please note that VHDL models are also available.

2.6.1 Cache Sizes

The instruction cache sizes available are 0–16 Kbytes, direct mapped or two-way set associative. The data cache sizes available are 0–16 Kbytes, direct mapped or two-way set associative. See Chapter 6, “CW4011 Caches,” and Appendix B, “Cache Sizing and Design Concerns,” for more information about cache sizing.

2.6.2 High-Performance Multiply Accumulate Unit

Each project may choose a high-performance multiply unit that provides base R3000 and R4000 multiply instructions (with similar performance) and `madd` and `msub` instructions. The high-performance multiplier is intended for applications with substantial multiply/accumulate performance needs. It includes a 32 x 32 pipelined array multiplier and a 64-bit accumulator that can retire a multiply or multiply/accumulate instruction every clock cycle with a latency of three clock cycles per result.

2.6.3 64-bit vs. 32-bit Memory Interface

The CW4011 BIU supports a 32-bit sizing interface for cost-sensitive designs or applications with low memory bandwidth. The BIU can be modified to present a 32-bit data bus instead of a 64-bit data bus.

2.6.4 Memory Management Unit

The CW4011 is designed to support the 32-bit addressing mode of the R4000 MMU. The TLB that is available in the base processor design contains up to 32 single-page entries. Each page can be individually specified to be 4 Kbytes or 16 Mbytes.

The CW4011 can support a simple MMU for designs that do not require a full MMU implementation. For designs with no TLB requirements, the TLB can also be removed to save silicon.

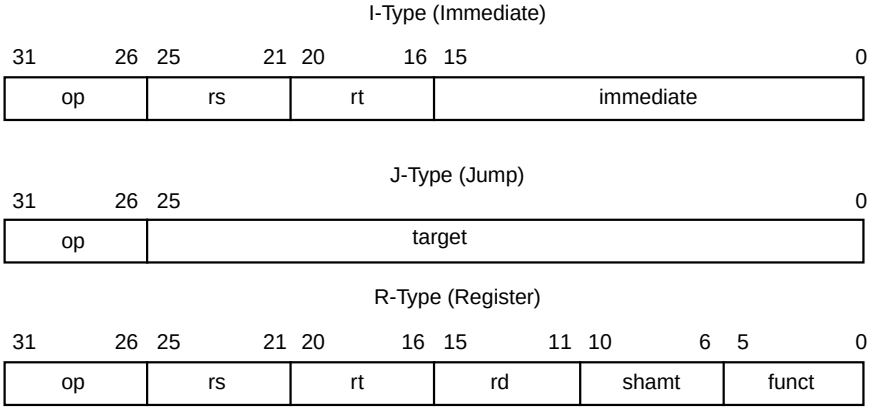
Chapter 3
Instruction Set

This chapter presents an overview of the MIPS R-series instructions and the instruction set extensions supported in the CW4011. This chapter contains the following sections:

3.1 Instruction Set Formats

Every R-series instruction consists of a single word (32 bits) aligned on a word boundary. As shown in Figure 3.1, there are three instruction formats: *I-type* (immediate), *J-type* (jump), and *R-type* (register). The restricted format approach simplifies instruction decoding. The compiler and assembler can synthesize more complicated (and less frequently used) operations and addressing modes.

Figure 3.1 Instruction Format



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

3.2 Load and Store Instructions

Load and store instructions are all I-type instructions and move data between memory and general purpose registers. The only addressing mode directly supported in the base R-series architecture is base register plus 16-bit signed immediate `offset`.

The MIPS II extensions add the Load Linked and Store Conditional instructions, which support multiple processors, and the Sync instruction, which synchronizes loads and stores. The CW4011 supports these instructions.

The load/store instruction operation code (opcode) determines the access type, which in turn indicates the size of the data item to be loaded or stored. Regardless of access type or byte-numbering order (big-endian or little-endian), the address specifies the byte that has the smallest byte address of all the bytes in the addressed field. For a

big-endian machine, the smallest byte is the leftmost byte; for a little-endian machine, it is the rightmost byte.

The bytes used within the addressed word can be determined directly from the access type and the two low-order bits of the address, as shown in Figure 3.2. Note that certain combinations of access type and low-order address bits can never occur; only the combinations shown in Figure 3.2 are allowed.

Figure 3.2 Byte Specifications for Loads/Stores

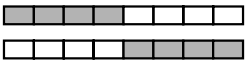
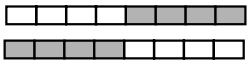
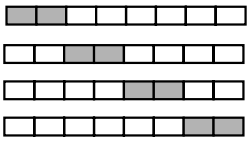
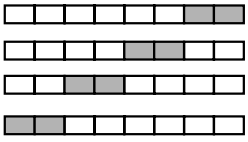
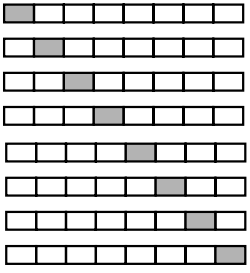
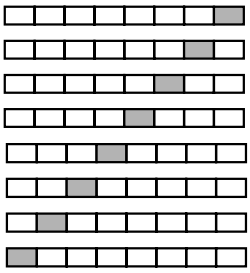
Access Type	Low-Order Address Bits A2 A1 A0	Data Bus: Big-Endian (63 … 0; Byte Numbers 0 1 2 3 4 5 6 7; MSB … LSB) — Bytes Accessed	Data Bus: Little-Endian (63 … 0; Byte Numbers 7 6 5 4 3 2 1 0; MSB … LSB) — Bytes Accessed
Doubleword	0 0 0		
Word	0 0 0 1 0 0		
Tribyte	0 0 0 0 0 1 1 0 0 1 0 1		
Halfword	0 0 0 0 1 0 1 0 0 1 1 0		
Byte	0 0 0 0 0 1 0 1 0 0 1 1 1 0 0 1 0 1 1 1 0 1 1 1		

Table 3.1 describes the load and store instructions supported by the CW4011. Instruction format is shown in courier; for example, `LB rt, offset(base)`.

Table 3.1 Load and Store Instruction Summary

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

(Sheet 1 of 2)

Table 3.1 Load and Store Instruction Summary (Cont.)

Instruction	Format and Description
Store Word	`SW rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Store contents of register `rt` at addressed location.
Store Word Left	`SWL rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift contents of register `rt` left so that the leftmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.
Store Word Right	`SWR rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Shift contents of register `rt` right so that the rightmost byte of the word is in the position of the addressed byte. Store word containing shifted bytes into word at addressed byte.
Load Linked	`LL rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address, and load the addressed word into register `rt`.
Store Conditional	`SC rt, offset(base)` Sign-extend the 16-bit `offset` and add to the contents of register base to form address. Conditionally store register `rt` at address, based on whether the load-link has been “broken.”
Sync	`SYNC` Complete all outstanding load and store instructions before allowing any new load or store instruction to start.
(Sheet 2 of 2)	

3.3 Computational Instructions

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

- Table 3.2 summarizes the ALU immediate instructions
- Table 3.3 summarizes the three-operand, register-type instructions
- Table 3.4 summarizes the shift instructions

- Table 3.5 summarizes the multiply/divide instructions
- Table 3.6 summarizes the computational CW4011 instruction extensions (CW4011 ISA)

Table 3.2 ALU Immediate Instruction Summary

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

Table 3.3 Three-Operand, Register-Type Instruction Summary

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

Table 3.4 Shift Instruction Summary

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

Table 3.5 Multiply/Divide Instruction Summary

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

Table 3.6 Computation Instruction Extensions Summary (CW4011 ISA)

Instruction	Format and Description
Add Circular Immediate	`ADDCIU rt, rs, immediate` The 16-bit `immediate` is sign-extended and added to the contents of general register `rs`, with the result masked by the value in CP0 register CMASK according to the formula: `rt = (rs`$_{31...cmask}$`||(rs+signextended_imed)`$_{cmask-1...0}$).
Find First Set Bit	`FFS rd, rs` Starting at the most-significant bit in register `rs`, find the first bit that is set to a one, and return the bit number in register `rd`. If no bit is set, return with all bits of `rd` set to 1.
Find First Clear Bit	`FFC rd, rs` Starting at the most-significant bit in register `rs`, find the first bit that is set to a zero, and return the bit number in register `rd`. If no bit is set, return with all bits of `rd` set to 1.
Minimum	`MIN rd, rs, rt` Compare the contents of registers `rs` and `rt` as two's complement values. The smaller value is stored in register `rd`.
Maximum	`MAX rd, rs, rt` Compare the contents of registers `rs` and `rt` as two's complement values. The larger value is stored in register `rd`.
Select and Shift Right	`SELSR rd, rs, rt` Using register `rs` and `rt` as a 64-bit register pair and the CP0 register Rotate as the shift count, shift the register pair `rs||rt` right the number of bits specified in Rotate, and place the least significant 32-bit value in result register `rd`.
Select and Shift Left	`SELSL rd, rs, rt` Using register `rs` and `rt` as a 64-bit register pair and the CP0 register Rotate as the shift count, shift the register pair `rs||rt` left the number of bits specified in Rotate, and place the most significant 32-bit value in result register `rd`.
Multiply/Add	`MADD rs, rt` Multiply contents of registers `rs` and `rt` as two's complement values. Add 64-bit results to contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Add Unsigned	`MADDU rs, rt` Multiply contents of registers `rs` and `rt` as unsigned values. Add 64-bit results to contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Subtract	`MSUB rs, rt` Multiply contents of registers `rs` and `rt` as two's complement values. Subtract 64-bit results from contents in special register pair HI/LO, and place results in HI and LO.
Multiply/Subtract Unsigned	`MSUBU rs, rt` Multiply contents of registers `rs` and `rt` as unsigned values. Subtract 64-bit results from contents in special register pair HI/LO, and place results in HI and LO.

Table 3.7 shows the execution time of the multiply/divide/accumulate type instructions.

Table 3.7 Execution Time of Multiply and Divide Instructions

Operation	R3000	CW33300	R4000	CW4011 High Speed
Multiply	12	1 + (bits/3)	10	3
Multiply/Add	na	na	na	3[1]
Divide	34	34	69	34/17[2]

1. For high-speed CW4011 multiply/add instructions, instructions can be pipelined for a throughput of one operation every clock cycle while the latency is three cycles. Pipelining the instructions accelerates calculations such as dot products and FIR filters that perform a series of multiplies/adds to compute a single result.
2. The divide time is shortened to 17 cycles if the divisor has less than 16 significant bits.

3.4 Jump and Branch Instructions

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

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

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

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

Table 3.8 summarizes the R-series jump instructions, Table 3.9 summarizes the branch instructions, and Table 3.10 summarizes the branch likely instructions.

Table 3.8 Jump Instruction Summary

Instruction	Format and Description
Jump	`J target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay.
Jump and Link	`JAL target` Shift 26-bit `target` address left two bits, combine with four high-order bits of PC, and jump to address with a one-instruction delay. Place address of instruction following delay slot in register 31 (link register).
Jump Register	`JR rs` Jump to address contained in register `rs` with a one-instruction delay.
Jump and Link Register	`JALR rs, rd` Jump to address contained in register `rs` with a one-instruction delay. Place address of instruction following delay slot in `rd`.

Table 3.9 Branch Instruction Summary

Instruction	Format and Description
Branch on Equal	`BEQ rs, rt, offset` Branch to target address[1] if register `rs` is equal to register `rt`.
Branch on Not Equal	`BNE rs, rt, offset` Branch to target address if register `rs` does not equal register `rt`.
Branch on Less than or Equal to Zero	`BLEZ rs, offset` Branch to target address if register `rs` is less than or equal to 0.
Branch on Greater Than Zero	`BGTZ rs, offset` Branch to target address if register `rs` is greater than 0.
Branch on Less Than Zero	`BLTZ rs, offset` Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero	`BGEZ rs, offset` Branch to target address if register `rs` is greater than or equal to 0.
Branch on Less Than Zero And Link	`BLTZAL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero And Link	`BGEZAL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is greater than or equal to 0.

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

Table 3.10 Branch Likely Instruction Summary (MIPS II ISA Extensions)

Instruction	Format and Description
Branch on Equal Likely	`BEQL rs, rt, offset` Branch to target address[1] if register `rs` is equal to register `rt`.
Branch on Not Equal Likely	`BNEL rs, rt, offset` Branch to target address if register `rs` does not equal register `rt`.
Branch on Less than or Equal to Zero Likely	`BLEZL rs, offset` Branch to target address if register `rs` is less than or equal to 0.
Branch on Greater Than Zero Likely	`BGTZL rs, offset` Branch to target address if register `rs` is greater than 0.
Branch on Less Than Zero Likely	`BLTZL rs, offset` Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero Likely	`BGEZL rs, offset` Branch to target address if register `rs` is greater than or equal to 0.
Branch on Less Than Zero And Link Likely	`BLTZALL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is less than 0.
Branch on Greater than or Equal to Zero And Link Likely	`BGEZALL rs, offset` Place address of instruction following delay slot in register 31 (link register). Branch to target address if register `rs` is greater than or equal to 0.

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

3.5 Trap Instructions

Trap instructions are part of the MIPS II instruction set and provide instructions that conditionally create an exception, based on the same conditions tested in the branch instructions. Table 3.11 provides a summary of MIPS II ISA extensions.

Table 3.11 Trap Instruction Summary (MIPS II ISA Extensions)

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

3.6 Special Instructions

Special instructions cause an unconditional branch to the general exception-handling vector. Special instructions are always R-type and are summarized in Table 3.12.

Table 3.12 Special Instruction Summary

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

3.7 Coprocessor Instructions

The CW4011 supports external (on-chip) coprocessors and implements the coprocessor instruction set. Please contact LSI Logic if your design needs more than one coprocessor. Coprocessor branch instructions are J-type. Table 3.13 summarizes the different coprocessor instructions.

Table 3.13 Coprocessor Instruction Summary

Instruction	Format and Description
Load Word to Coprocessor z	`LWCz rt, offset(base)` Extends the sign of the 16-bit `offset` and adds the `offset` to the contents of the general register base to form a 32-bit unsigned effective address. The word at the memory location specified is loaded into coprocessor register `rt` of the coprocessor unit `z`.
Store Word from Coprocessor z	`SWCz rt, offset(base)` Extends the sign of the 16-bit `offset` and adds the `offset` to the contents of the general register base to form a 32-bit unsigned effective address. The contents of coprocessor register `rt` of the coprocessor unit `z` are stored at the address specified by the 32-bit unsigned effective address.
Move To Coprocessor z	`MTCz rt, rd` Loads the contents of general register `rt` into the `rd` register of coprocessor unit `z`.
Move From Coprocessor z	`MFCz rt, rd` Loads the contents of the `rd` register of coprocessor unit `z` into general register `rt`.
Move Control to Coprocessor z	`CTCz rt, rd` Loads the contents of general register `rt` into the control register `rd` of coprocessor unit `z`.
Move Control From Coprocessor z	`CFCz rt, rd` Loads the contents of the control register `rd` of coprocessor unit `z` into general register `rt`.
Coprocessor Operation	`COPz cofun` Initiates a coprocessor operation that may specify and reference the coprocessor's internal registers or change the state of the coprocessor's condition line, but does not change the state within the processor or the cache memory.
Branch on Coprocessor z True (Likely)	`BCzT offset, (BCzTL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if coprocessor z's condition line is true. In the case of branch likely, the delay slot instruction is not executed when the branch is not taken.
Branch on Coprocessor z False (Likely)	`BCzF offset, (BCzFL offset)` Compute a branch target address by adding address of instruction to the 16-bit `offset` (shifted left two bits and sign-extended to 32 bits). Branch to the target address (with a delay of one instruction) if coprocessor z's condition line is false. In the case of branch likely, the delay slot instruction is not executed when the branch is not taken.

3.8 System Control Coprocessor (CP0) Instructions

Coprocessor 0 instructions perform operations on the system control coprocessor (CP0) registers to manipulate the memory management and exception-handling facilities of the processor. Table 3.14 summarizes the CP0 instructions.

If the TLB is removed, the TLB instructions (TLBR, TLBWI, TLBWR, TLBP) cause an RI (Reserved Instruction) exception. If the CW4011 is in R3000 compatibility mode, the ERET (Exception Returned) instruction is unavailable, and this causes an RI exception. Conversely, if the CW4011 is in R4000 mode, the RFE (Restore From Exception) instruction is unavailable, and this causes an RI exception.

Table 3.14 CP0 Instruction Summary

Instruction	Format and Description
Move To CP0	`MTC0 rt, rd` Loads contents of CPU register `rt` into CP0 register `rd`.
Move From CP0	`MFC0 rt, rd` Loads contents of CP0 register `rd` into CPU register `rt`.
Read Indexed TLB Entry[1]	`TLBR` Loads EntryHi and EntryLo with the TLB entry pointed to by the Index register.
Write Indexed TLB Entry[1]	`TLBWI` Loads TLB entry pointed to by the Index register with the contents of the EntryHi and EntryLo registers.
Write Random TLB Entry[1]	`TLBWR` Loads TLB entry pointed to by the Random register with the contents of the EntryHi and EntryLo registers.
Probe TLB for Matching Entry[1]	`TLBP` Loads the Index register with the address of the TLB entry whose contents match the EntryHi and EntryLo registers. If no TLB entry matches, set the high-order bit of the Index register.

(Sheet 1 of 2)

Table 3.14 CP0 Instruction Summary (Cont.)

Instruction	Format and Description
Exception Return[2]	ERET (R4000 mode) Loads the PC from ErrorEPC (SR2 = 1: Error Exception) or EPC (SR2 = 0: Exception) and clear ERL bit (SR2 = 1) or EXL bit (SR2 = 0) in the Status Register. SR2 is Status register bit 2.
Restore From Exception[2]	RFE (R3000 mode) Restores previous interrupt mask and mode bits of the Status register into current status bits. Restore old status bits into previous status bits.
Wait for Interrupt	WAITI Stops execution of instructions and places the processor into a power save (stall) condition until a hardware interrupt, NMI, or reset is received.
(Sheet 2 of 2)	

1. If there is no MMU installed, any of these instructions can cause a reserved instruction exception.
2. Only one of these instructions is legal at any one time. The one that is not legal causes a reserved instruction exception.

3.9 Cache Maintenance Instructions

Cache Maintenance instructions are always I-type. Table 3.15 summarizes these instructions.

Table 3.15 Cache Maintenance Instruction Summary

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

3.10 CW4011 Instruction Set Extensions

This section defines the CW4011 instruction set extensions. Table 3.16 lists all the extensions and the page where a description can be found.

Table 3.16 CW4011 Instruction Set Extensions

Extension	Page	Extension	Page
ADDCIU	3-22	MAX	3-30
FFC	3-23	MIN	3-31
FFS	3-24	MSUB	3-32
FLUSHD	3-25	MSUBU	3-33
FLUSHI	3-26	SELSL	3-34
FLUSHID	3-27	SELSR	3-35
MADD	3-28	WAITI	3-36
MADDU	3-29	WB	3-37

ADDCIU **Add Circular Immediate**

Format

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

Syntax **`ADDCIU rt, rs, immediate`**

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

The carries resulting from the addition of the sign-extended `offset` are not propagated into the final result beyond bit [CMask-1].

Operation

```
T: sign_extend_immed = (immediate15)16 || immediate15..0
   GPR[rt] = GPR[rs]31..cmask || (GPR[rs] +
    sign_extend_immed)cmask-1..0
```

Exceptions None

FFC **Find First Clear Bit**

Format

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

Syntax **`FFC rd, rs`**

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

Exceptions None

FFS **Find First Set Bit**

Format

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

Syntax **`FFS rd, rs`**

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

Exceptions None

FLUSHD **FLUSH Data Cache**

Format

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

Syntax **FLUSHD**

Description FLUSHD flushes all D-cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

FLUSHI **FLUSH Instruction Cache**

Format

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

Syntax **FLUSHI**

Description FLUSHI flushes all I-cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

FLUSHID **FLUSH Instruction and Data Cache**

Format

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

Syntax **`FLUSHID`**

Description FLUSHID flushes all D-cache and I-cache lines and causes stall cycles for 256 clocks, regardless of the cache size.

Exceptions None

MADD **Multiply/Add**

Format

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

Syntax **`MADD rs, rt`**

Description The contents of general register `rs` and the contents of general register `rt` are multiplied. Both operands are treated as 32-bit two's complement values. When the operation is completed, the doubleword result is added to special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the Configuration and Cache Control (CCC) register.

MADD executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.18 on page 3-39.

Operation

```
T:      t <- (HI || LO) + (GPR[rs] * GPR[rt])
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MADDU **Multiply/Add Unsigned**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MADDU
000000	rs	rt	0	00000	011101

Syntax **`MADDU rs, rt`**

Description The contents of general register `rs` and the contents of general register `rt` are multiplied with both operands treated as 32-bit unsigned values. When the operation is completed, the doubleword result is added to special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.18 on page 3-39.

Operation

```
T:      t <- (HI || LO) + ((0||GPR[rs]) * (0||GPR[rt]))
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MAX **Maximum**

Format

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

Syntax **`MAX rd, rs, rt`**

Description The source operands `rs` and `rt` are compared as two's complement values. The larger value is stored in the `rd` register.

Operation

```
T:      if GPR[rs]>GPR[rt] then
              GPR[rd]<-GPR[rs]
        else
              GPR[rd]<-GPR[rt]
        endif
```

Exceptions None

MIN **Minimum**

Format

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

Syntax **`MIN rd, rs, rt`**

Description The source operands `rs` and `rt` are compared as two's complement values. The smaller value is stored in the `rd` register.

Operation

```
T:      if GPR[rs]<GPR[rt] then
              GPR[rd]<-GPR[rs]
        else
              GPR[rd]<-GPR[rt]
        endif
```

Exceptions None

MSUB **Multiply/Subtract**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MSUB
000000	rs	rt	0	00000	011110

Syntax **`MSUB rs, rt`**

Description The contents of general register `rs` and `rt` are multiplied and both operands are treated as 32-bit two's complement values. When the operation is complete, the doubleword result is subtracted from special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.18 on page 3-39.

Operation

```
T:      t <- (HI || LO) - (GPR[rs] * GPR[rt])
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

MSUBU **Multiply/Subtract Unsigned**

Format

31 26	25 21	20 16	15 11	10 6	5 0
SPECIAL	rs	rt	0	0	MSUBU
000000	rs	rt	0	00000	011111

Syntax **`MSUBU rs, rt`**

Description The contents of general register `rs` and `rt` are multiplied and both operands are treated as 32-bit unsigned values. When the operation is completed, the doubleword result is subtracted from special register pair HI/LO.

No overflow exception occurs under any circumstances.

This instruction is only available when the chip has multiplier-accumulator module hardware and MAD/MUL are set to one in the CCC register.

The instruction executes in multiple cycles, depending on the number of significant bits in the operands. Refer to Table 3.18 on page 3-39.

Operation

```
T:      t <- (HI || LO) - ((0||GPR[rs]) * (0||GPR[rt]))
        LO <- t31..0 , HI <- t63..32
```

Exceptions None

SELSL **Select and Shift Left**

Format

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

Syntax **SELSL rd, rs, rt**

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

Operation

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

Exceptions None

SELSR **Select and Shift Right**

Format

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

Syntax **SELSR rd, rs, rt**

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

Operation

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

Exceptions None

WAITI **Wait for Interrupt**

Format

31 26	25 21	20 16	15 11	10 6	5 0
COP0		0	0	0	WAITI
010000	10000	00000	00000	00000	100000

Syntax **`WAITI`**

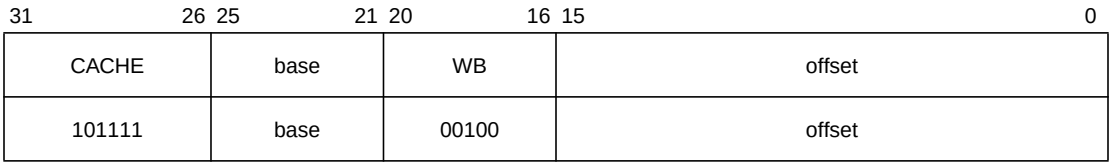
Description When this instruction is executed, the main processor clock stops and execution of instructions is halted. Execution resumes when a hardware interrupt, NMI, or reset exception is received. While it is in wait mode, the processor is in a power saving mode, using very little current because the clock is turned off to most of the circuitry.

WAITI must be followed by two or more NOP instructions, otherwise, the results may be undefined. Refer to Appendix C, “Programmer’s Notes,” for further information.

Exceptions None

WB **WriteBack**

Format



Syntax **`WB offset(base)`**

Description Eight words of the D-cache line addressed by `offset +GPR[base]` are written back to memory if the line is dirty. Upper bits of `offset +GPR[base]` are ignored.

Exceptions None

3.11 CPU Instruction Opcode Bit Encoding

Tables 3.17–3.23 show the opcode bit encoding for CW4011 instructions. The following key applies to operation codes referenced in the table:

***rxf1** Cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture.

***rxf2** Cause reserved instruction exceptions in all current implementations and are reserved for future versions of the architecture. *rxf2 is separated from other reserved instructions for COPz. These are not detected as reserved instruction codes that cause an exception on the R3000. The R4000 detects them.

***rx40** Cause a reserved instruction exception on R4000 and CW4011 processors (when in R4000 mode). They are used as a Restore From Exception (RFE) instruction on the R3000, LR33000, LR33300, and CW4011 in R3000 mode.

***rx64** Cause a reserved instruction exception. They are 64-bit instructions on R4000.

***nrx** Invalid but do not cause reserved instruction exceptions in CW4011 implementations.

x1 Originally, extended instructions in CW4011 implementations. They are reserved instructions that cause an exception on R4000.

x2 The operation code CACHE marked with x2 is valid only for CW4011 processors with CP0 enabled and causes a reserved instruction exception with CP0 disabled. Bits [20:16] are sub-opcodes. They are instructions for cache maintenance, and the functions are not compatible with R4000. Recommended mnemonics are FLUSHI, FLUSHD, FLUSHID, and WB `offset`(base). Undefined opcodes of CACHE instruction do not cause reserved instruction exception in CW4011 implementations.

x3 Originally, extended instructions in CW4011 implementations. They are used for 64-bit multiply and divide instructions on R4000. If the MUL bit or MAD bit in the CCC register is zero, they cause a reserved instruction exception. The CCC register is described in detail in Section 4.3.10, “Configuration and Cache Control (CCC) Register,” on page 4-22.

x4 Cause a reserved instruction exception if the MUL bit in the CCC register is zero.

x5 The operation code ERET marked with x5 is valid only on the R4000 and CW4011 in R4000 mode.

x6 Coprocessor 3 instructions, which are not available on R4000. They are available on the R3000 and CW4011.

Table 3.17 CW4011 Opcode Bit Encoding

	[28:26]			Opcode				
[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	COP3[x6]	BEQL	BNEL	BLEZL	BGTZL
3	*rx64	*rx64	*rx64	*rx64	ADDCIU[x1]	*rxf1	*rxf1	*rxf1
4	LB	LH	LWL	LW	LBU	LHU	LWR	*rx64
5	SB	SH	SWL	SW	*rx64	*rx64	SWR	CACHE[x2]
6	LL	LWC1	LWC2	LWC3[x6]	*rx64	*rx64	*rx64	*rx64
7	SC	SWC1	SWC2	SWC3[x6]	*rx64	*rx64	*rx64	*rx64

Table 3.18 SPECIAL Opcode Bit Encoding

	[2:0]			SPECIAL Function				
[5:3]	0	1	2	3	4	5	6	7
0	SLL	SELSR[x1]	SRL	SRA	SLLV	SELSL[x1]	SRLV	SRAV
1	JR	JALR	FFS[x1]	FFC[x1]	SYSCALL	BREAK	*rxf1	SYNC
2	MFHI[x4]	MTHI[x4]	MFLO[x4]	MTLO[x4]	*rx64	*rxf1	*rx64	*rx64
3	MULT[x4]	MULTU[x4]	DIV[x4]	DIVU[x4]	MADD[x3]	MADDU[x3]	MSUB[x3]	MSUBU[x3]
4	ADD	ADDU	SUB	SUBU	AND	OR	XOR	NOR
5	MIN[x1]	MAX[x1]	SLT	SLTU	*rx64	*rx64	*rx64	*rx64
6	TGE	TGEU	TLT	TLTU	TEQ	*rxf1	TNE	*rxf1
7	*rx64	*rxf1	*rx64	*rx64	*rx64	*rxf1	*rx64	*rx64

Table 3.19 REGIMM Opcode rt Bit Encoding

	[18:16]			REGIMM rt				
[20:19]	0	1	2	3	4	5	6	7
0	BLTZ	BGEZ	BLTZL	BGEZL	*rxf1	*rxf1	*rxf1	*rxf1
1	TGEI	TGEIU	TLTI	TLTIU	TEQI	*rxf1	TNEI	*rxf1
2	BLTZAL	BGEZAL	BLTZALL	BGEZALL	*rxf1	*rxf1	*rxf1	*rxf1
3	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1	*rxf1

Table 3.20 CACHE^{x2} Opcode rt Bit Encoding

	[18:16]			CACHE^{x2} rt				
[20:19]	0	1	2	3	4	5	6	7
0	*nrx	FLUSHI^{x2}	FLUSHD^{x2}	FLUSHID^{x2}	WB^{x2}	*nrx	*nrx	*nrx
1	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
2	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
3	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx

Table 3.21 COPz rs Opcode Bit Encoding

	[23:21]			COPz rs				
[25:24]	0	1	2	3	4	5	6	7
0	MFCz	*rx64	CFCz	*rxf2	MTCz	*rx64	CTCz	*rxf2
1	BC	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
2	COPz (Coprocessor defined instructions)							
3								

Table 3.22 COPz rt Opcode Bit Encoding

	[18:16]			COPz rt			
[20:19]	0	1	2	3	4	5	6	7
0	BCF	BCT	BCFL	BCTL	*rxf2	*rxf2	*rxf2	*rxf2
1	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2
3	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2	*rxf2

Table 3.23 CP0 Opcode Bit Encoding

	[2:0]			CP0 Function				
[5:3]	0	1	2	3	4	5	6	7
0	*nrx	TLBR	TLBWI	*nrx	*nrx	*nrx	TLBWR	*nrx
1	TLBP	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
2	RFE^{rx40}	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
3	ERET^{x5}	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
4	WAITI^{x1}	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
5	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
6	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx
7	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx	*nrx

Chapter 4
CW4011 Exception Processing

This chapter describes the CW4011 system coprocessor, Coprocessor 0 (CP0), and explains how the CW4011 handles exception processing. The chapter is divided into the following sections:

4.1 Overview

When the CW4011 detects an exception, it suspends the normal sequence of instruction execution, exits from User mode, and enters Kernel mode where it can handle exceptions. The CW4011 reverts to Kernel mode, regardless of the mode at the time of the exception. The processor then disables interrupts and forces a software handler located at a fixed address in memory to be executed. The handler saves the context of the processor. The context must be restored when the exception has been handled. Section 5.2.1, “Operating Modes,” provides more information on this subject.

When an exception occurs, the CP0 loads the Exception Program Counter (EPC) with a restart location where execution may resume after the exception has been serviced. The restart location in the EPC is the address of the instruction that caused the exception or, if the instruction was executing in a Branch Delay slot, the address of the branch instruction immediately preceding the delay slot. The instruction causing the exception and all the instructions following in the pipeline are aborted. They will be refetched after return from the exception.

This chapter describes the events that can initiate exception processing. Table 4.1 summarizes these events.

Table 4.1 CW4011 Exceptions

Exception	Cause
Cold Reset	Deassertion of the CW4011 cold reset signal, CRESETn.
Warm Reset	Deassertion of the CW4011 warm reset signal, WRESETn.
Nonmaskable Interrupt	Assertion of the nonmaskable interrupt signal, NMIn.
Debug	Detection of a program counter breakpoint, data address breakpoint, or trace event. Not supported in standard R3000 and R4000 processors.
Address Error	Either an attempt to load, fetch, or store a word not aligned on a word boundary, or an attempt to load or store a halfword not aligned on a halfword boundary. References to an address for which the most significant bit was set while in the CW4011 was in User mode may also cause an address error.
TLB Refill	There is no TLB entry to match a reference to a mapped address space.
TLB Entry Invalid	A virtual address reference matches a TLB entry that is marked invalid.
TLB Modified	A store operation's virtual address reference matches a TLB entry that is marked valid but is not dirty/writable.
Bus Error	Assertion of the CW4011 external bus error signal, SCBERRn.
Integer Overflow	Two's complement overflow during an add or subtract.
Trap	One of the trap instructions results in a "true" condition.
System Call	An attempt to execute the SYSCALL instruction.
Breakpoint	An attempt to execute the BREAK instruction.
Reserved Instruction	Execution of an instruction with an undefined or reserved major operation code (bits [31:26]), or a SPECIAL instruction whose minor operation code (bits [5:0]) is undefined.
Coprocessor Unusable	Execution of a coprocessor instruction where the Cu (coprocessor usable) bit is not set for the target coprocessor.
Floating Point	Available for use by an external floating-point coprocessor.
(Sheet 1 of 2)	

Table 4.1 CW4011 Exceptions (Cont.)

Exception	Cause
Interrupt	Assertion of one of the CW4011's six hardware interrupt inputs, or the setting of one of the two software interrupt bits in the Cause register. Interrupts must be enabled.
External Vectored Interrupt	Assertion of the CW4011 EXViNTn input. Not supported in R3000 and R4000 processors.
(Sheet 2 of 2)	

4.2 R3000 Exception Compatibility Mode

Although the CW4011 processor is based on the MIPS R4000 architecture, an R3000-style exception processing capability has been added. This facility allows you to configure CP0 exception processing in such a way that existing R3000 exception handling code can be run on the CW4011 processor with little or no modification to the code.

R3000 compatibility mode is under the control of the compatibility bit (bit 24) of the Configuration and Cache Control (CCC) register, discussed in Section 4.3.10. The compatibility bit is reset to zero (R4000 mode) when a cold reset exception occurs. If R3000 mode operation is desired, bit 24 should be set to one as part of the cold reset handler. Once it has been placed in R3000 mode, the processor should only be switched back to R4000 mode by another cold reset. When R3000 mode is enabled, the behavior of the following areas is affected:

- Status Register

  The lower six bits of the Status register are redefined to implement the Kernel/User mode and interrupt enable stack as defined by the R3000 architecture. The Status register is discussed in detail in Section 4.3.6, "Status Register."

- Exception Handling Vectors

  The exception handling vectors (base and offset) are remapped to those specified by the R3000 architecture. The exception vectors are discussed in detail in Section 4.4.3, "Exception Vector Locations."

- Exception Return (RFE vs. ERET)

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

The following sections provide more detail on CW4011 exception handling. Where appropriate, the differences between standard operation R4000 and R3000 compatibility mode are noted. In all other cases, operation is identical.

4.3 Exception Handling Registers

This section describes the CP0 registers used in exception processing. Software examines these registers during exception processing to determine the cause of an exception and the state of the CPU at the time of the exception. Each of the registers is listed in Table 4.2 and described in detail in the sections that follow.

Table 4.2 CP0 Exception Processing Registers

Register Name	CP0 Register Number	Reference Page
Context	4	4-5
Debug Control and Status (DCS)	7	4-7
Bad Virtual Address (BadVAddr)	8	4-9
Count	9	4-9
Compare	11	4-9
Status	12	4-10
Cause	13	4-18
Exception Program Counter (EPC)	14	4-20
Processor Revision Identifier (PRId)	15	4-20

(Sheet 1 of 2)

Table 4.2 CP0 Exception Processing Registers (Cont.)

Register Name	CP0 Register Number	Reference Page
Configuration and Cache Control (CCC)	16	4-22
Load Linked Address (LLAdr)	17	4-26
Breakpoint Program Counter (BPC)	18	4-27
Breakpoint Data Address (BDA)	19	4-27
Breakpoint PC Mask (BPCM)	20	4-27
Breakpoint Data Address Mask (BDAM)	21	4-28
Rotate	23	4-28
Circular Mask (CMask)	24	4-29
Error Exception Program Counter (Error EPC)	30	4-30
(Sheet 2 of 2)		

Two other CP0 registers that are part of the virtual memory management system and contain important information about exception handling are the Index register (CP0 register 0), described in Section 5.3.2.4, “Index Register,” and the Random register (CP0 register 1), described in Section 5.3.2.5, “Random Register.”

You can use the MTC0 (Move To Coprocessor 0) instruction to set the bits in the registers, and MTF0 (Move From Coprocessor 0) to read the contents of the registers.

4.3.1 Context Register

The Context register is a read/write register containing a 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, operating system software handles the miss by loading the TLB with the missing translation from the PTE array.

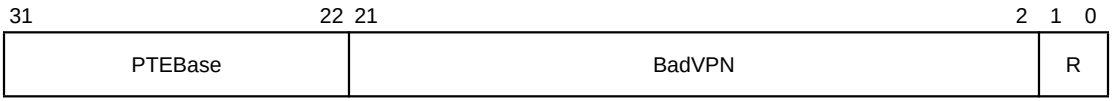
The BadVPN field is not writable. It contains the VPN of the most recently translated virtual address that did not have a valid translation (TLBL or TLBS). The PTEBase field is both writable and readable, and

indicates the base address of the PTE table of the current user address space.

The Context register duplicates some of the information provided in the BadVAddr register, but the information is in a form that is more useful for a software TLB exception handler.

The Context register can be used by the operating system to hold a pointer into the PTE array. The operating system sets the PTE base field register, as needed. Normally, the operating system uses the Context register to address the current page map, which resides in the kernel-mapped segment *kseg2*. The register is included solely for the use of the operating system. Figure 4.1 shows the format of the Context register.

Figure 4.1 Context Register



PTEBase **Page Table Entry Base** **[31:22]**
This field is the Operating System Pointer. It points to the PTE in memory.

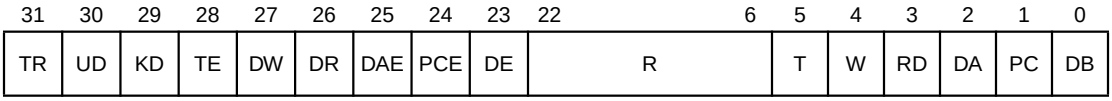
BadVPN **Bad Virtual Page Number** **[21:2]**
This field contains bits [31:12] of the most recently translated virtual address that did not have a valid translation. This format provides a table of four-byte PTEs for a page size of 4 Kbytes. For other PTE and page sizes, shifting and masking bits [21:2] produces an appropriate address.

R **Reserved** **[1:0]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

4.3.2 Debug Control and Status (DCS) Register

The DCS register contains the enable and status bits for the CW4011 debug facility. All bits have read/write access. Figure 4.2 shows the format of the DCS register.

Figure 4.2 DCS Register



TR **Trap** **31**
This is the trap enable bit. Setting it to one traps debug events to the debug exception vector. Clearing it to zero disables the trap. However, the status bits (UD, KD, etc.) are updated with status debug event information even when the bit is cleared.

UD **User Mode Debug Event** **30**
This bit is set to one to enable detection of a debug event when the CW4011 is operating in User mode.

KD **Kernel Mode Debug Event** **29**
This bit is set to one to enable detection of a debug event when the CW4011 is operating in Kernel mode.

TE **Trace Event** **28**
This bit is set to one to enable detection of a trace event (nonsequential fetch operation).

DW **Data Write** **27**
This bit is set to one to enable detection of a data write event as defined by the BDA and BDAM registers. The bit is used in conjunction with DAE.

DR **Data Read** **26**
This bit is set to one to enable detection of a data read event as defined by the BDA and BDAM registers. The bit is used in conjunction with DAE.

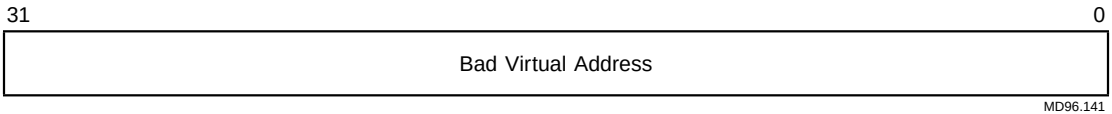
DAE **Detect BDA Event** **25**
This bit is set to one to enable detection of a BDA debug event.

PCE	**Program Counter Breakpoint Event** This bit is set to one to enable detection of a program counter breakpoint event as defined by the BPC and BPCM registers.	**24**
DE	**Debug Enable** This bit is set to one to enable the debug facility. Clearing the bit disables the debug facility.	**23**
R	**Reserved** These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future software revisions.	**[22:6]**
T	**Trace Status** The core sets T to one when it detects a trace condition.	**5**
W	**Write Status** The core sets W to one when it detects a write reference to the address specified in the Breakpoint Address register.	**4**
RD	**Read Status** The core sets RD to one when it detects a read reference to the address specified in the Breakpoint Data Address register.	**3**
DA	**DAE Debug Condition Status** The core sets DA to one when it detects a data address debug condition.	**2**
PC	**PCE Debug Condition Status** The core sets PC to one when it detects a program counter debug condition.	**1**
DB	**Debug Detected Status** The core sets DB to one when it detects any debug condition.	**0**

4.3.3 Bad Virtual Address (BadVAddr) Register

The BadVAddr register is a read-only register that holds the 32-bit failing virtual address for address error (AdEL, AdES) and TLB translation (TLBL, TLBS, Mod) exceptions. Figure 4.3 shows the format of the BadVAddr register.

Figure 4.3 BadVAddr Register



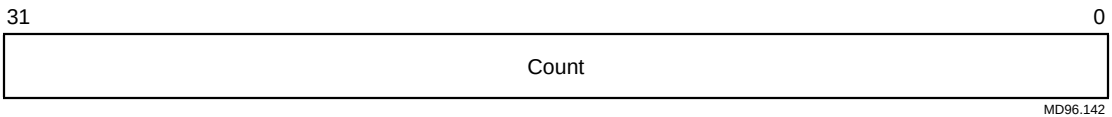
4.3.4 Count Register

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

The Count register is a read/write register—it can be written for diagnostic purposes or for system initialization to synchronize two processors operating in lock step.

Figure 4.4 shows the format of the Count register.

Figure 4.4 Count Register

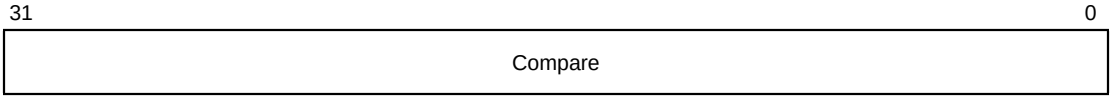


4.3.5 Compare Register

The Compare register implements a timer service (see also the Count register) that maintains a stable value and is not automatically updated by core events. When the timer facility is enabled and the value of the Count register equals the value of the Compare register, interrupt bit IP[7] in the Cause register is set. This causes an interrupt on the next execution cycle when the interrupt is enabled. Writing a value to the Compare register clears the timer interrupt.

For diagnostic purposes, the Compare register is a read/write register. In normal operation, the Compare register is only written. Figure 4.5 shows the format of the Compare register.

Figure 4.5 Compare Register



4.3.6 Status Register

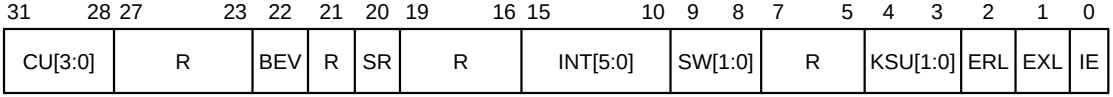
The Status register is a read/write register that contains the operating mode, interrupt enabling, and the diagnostic states of the processor. The format of the Status register is slightly different when the CW4011 is operating in R4000 mode from when it is in R3000 mode. Section 4.3.6.1, "R4000 Mode Operation," describes the format for R4000 mode operation and Section 4.3.6.2, "R3000 Mode Operation," describes the format for R3000 mode operation.

4.3.6.1 R4000 Mode Operation

The format of the R4000 version of the Status register (CCC24 = 0) is shown in Figure 4.6. Following the figure are bit-field descriptions and information on these R4000 operations:

- Interrupt Enable
- Processor Modes
- Kernel Address Space Accesses
- User Address Space Accesses
- Cold Reset
- Warm Reset

Figure 4.6 Status Register (R4000 Mode)



CU[3:0] **Coprocessor Usability Bits** **[31:28]**

The software uses this field to control accesses to the coprocessors. When the bit is set to one the corresponding coprocessor is usable:

CU[3:0]	Coprocessor
3	3
2	2
1	1
0	0

Please note that CP0 is always available in Kernel mode regardless of the CU[0] setting.

R **Reserved** **[27:23, 21, 19:16, 7:5]**

These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

BEV **Bootstrap Exception Vector** **22**

This bit controls the location of the TLB refill and the general exception vectors. Setting the bit to one indicates a bootstrap operation and bootstrap vector locations are used. When the bit is cleared to zero, normal exception vectors are used.

SR **Soft Reset** **20**

When a warm reset or a nonmaskable interrupt occurs, the core sets SR to one.

INT[5:0] **Interrupt Mask** **[15:10]**

This field is a six-bit [5:0] hardware interrupt mask. Setting a bit to one enables the corresponding hardware interrupt. For example, setting bit 5 enables hardware interrupt 5.

SW[1:0] **Software Interrupt Mask** **[9:8]**
This field is a two-bit [1:0] software interrupt mask. Setting a bit to one enables the corresponding software interrupt.

KSU[1:0] **Kernel/User Mode** **[4:3]**
This field determines the base operating mode of the CW4011 core as follows:

Bit	Base Mode
00	Kernel
10	User

All other settings are reserved.

ERL **Error Level** **2**
This bit determines the error level of the CW4011. When it is set to one, the level is Error. When it is cleared to zero, the level is Normal.

EXL **Exception Level** **1**
This bit determines the exception level of the CW4011. When it is set to one, the level is Exception. When it is cleared to zero, the level is Normal.

IE **Interrupt Enable** **0**
Setting this bit to one enables interrupts. Clearing it to zero disables interrupts.

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

- IE is set to one
- EXL is cleared to zero
- ERL is cleared to zero

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

Processor Modes – The setting of the KSU bits, in conjunction with the settings of the EXL and ERL bits, defines the CW4011 processor modes as follows:

- The processor is in User mode when KSU is equal to 0b10, and EXL and ERL are cleared to zero.
- The processor is in Kernel mode under any one of the following conditions:
  - KSU is equal to 0b00
  - EXL is set to one
  - ERL is set to one

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

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

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

- ERL and BEV bits are set to one
- CU[3:0] and SR bits are set to zero

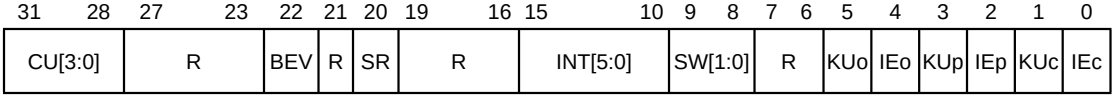
Warm Reset – The contents of the Status register are unchanged by warm reset, except for these bits: ERL, BEV, and SR bits are set to one.

4.3.6.2 R3000 Mode Operation

The format of the R3000 version of the Status register (CCC24 = 1) is shown in Figure 4.7. Following the figure are bit-field descriptions and descriptions of these R3000 operations:

- Interrupt Enable
- Processor Modes
- Kernel Address Space Accesses
- User Address Space Accesses
- Cold Reset
- Warm Reset
- Mode Bits and Exception Processing

Figure 4.7 Status Register (R3000 Mode)



CU[3:0] **Coprocessor Usability Bits** **[31:28]**

The software uses this field to control accesses to the coprocessors. When the bit is set to one the corresponding coprocessor is usable, as shown below:

CU[3:0]	Coprocessor
3	3
2	2
1	1
0	0

Please note that CP0 is always available in Kernel mode regardless of the CU[0] setting.

R **Reserved** **[27:23, 21, 19:16, 7:6]**

These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

BEV **Bootstrap Exception Vector** **22**

This bit controls the location of the TLB refill and the general exception vectors. Setting the bit to one implements a bootstrap operation and bootstrap vector locations are used. When the bit is cleared to zero, normal exception vectors are used.

SR **Soft Reset** **20**

When either a warm reset or a nonmaskable interrupt occurs, the core sets SR to one.

INT[5:0] **Interrupt Mask** **[15:10]**

This field is a six-bit [5:0] hardware interrupt mask. Setting a bit to one enables the corresponding hardware interrupt. For example, setting bit 5 to one enables hardware interrupt 5.

SW[1:0] **Software Interrupt Mask** **[9:8]**
This field is a two-bit [1:0] software interrupt mask. Setting a bit to one enables the corresponding software interrupt.

KUo **Kernel/User Mode, Old** **5**
This bit shows the old base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

IEo **Interrupt Enable, Old** **4**
This bit shows the old interrupt enable setting. Setting it to one indicates that interrupts are enabled. Clearing the bit to zero indicates that interrupts are disabled. The bit is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

KUp **Kernel/User Mode, Previous** **3**
This bit shows the previous base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

IEp **Interrupt Enable, Previous** **2**
This bit shows the previous interrupt enable setting. Setting it to one indicates that interrupts are enabled. Clearing the bit to zero indicates that interrupts are disabled. The bit is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

KUc **Kernel/User Mode, Current** **1**
This bit shows the current base operating mode of the CW4011 core. Setting it to one indicates User mode. Clearing the bit to zero indicates Kernel mode. The bit is part of a three-bit stack that indicates old, previous, and current modes.

IEc **Interrupt Enable, Current** **0**
This bit shows the old interrupt enable setting. Setting it to one indicates that interrupts are enabled. Clearing the bit to zero indicates that interrupts are disabled. The bit

is part of a three-bit stack that indicates old, previous, and current interrupt enable settings.

Interrupt Enable – Interrupts are enabled when IEc is set to one. In this case, interrupts are recognized according to the setting of the INT and SW masks.

Processor Modes – CW4011 processor modes are defined by the setting of the KUc bit:

- The processor is in User mode when KUc is set to one.
- The processor is in Kernel mode when KUc is cleared to zero.

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

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

Cold Reset – The CW4011 processor enters R4000 mode upon cold reset. Refer to “Cold Reset” on page 4-13 for the initial Status register settings for this mode. To enter R3000 mode, set bit 24 of the Configuration and Cache Control (CCC) register to one as part of the cold reset handler.

Upon entering R3000 mode after a cold reset, the contents of the Status register are undefined except for the following bits:

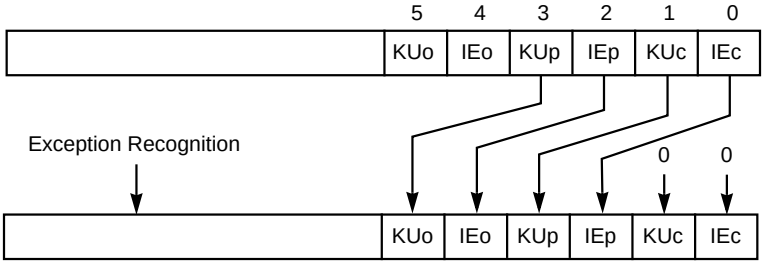
- The BEV bit is set to one.
- The CU[3:0], KUc, KUo, KUp, IEc , IEo, IEp, and SR bits are cleared to zero.

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

- The BEV and SR bits are set to one.
- The KU and IE bits are pushed deeper into the stack and KUc and IEc are cleared to zero, for example:
  KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0.

Mode Bits and Exception Processing – Figure 4.8 shows how the CW4011 core manipulates the Status register during exception recognition.

Figure 4.8 Status Register and Exception Recognition



KUc	Current Kernel/User mode bit
IEc	Current Interrupt Enable mode bit
KUp	Previous Kernel/User mode bit
IEp	Previous Interrupt Enable mode bit
KUo	Old Kernel mode bit
IEo	Old Interrupt Enable mode bit

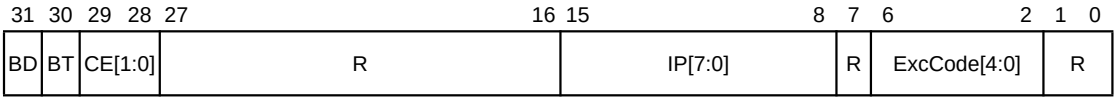
When the CW4011 recognizes an exception, it saves the current Kernel/User mode bit (KUc) and the current interrupt enable bit (IEc) in the previous Kernel/User mode bit (KUp) and previous interrupt enable bit (IEp), respectively. The previous bits are saved in the old bits, and the current bits are cleared to zero. The process is shown in the following example: KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0.

When the CW4011 executes a Return From Exception (RFE) instruction, the values are popped off the stack, KUc and IEc are reset to their previous values, for example: KUc/IEc ← KUp/IEp ← KUo/IEo.

4.3.7 Cause Register

The Cause register is a read/write register. The contents of this register provide information about the most recent exception. The format of the Cause register is shown in Figure 4.9. All bits in the register, with the exception of IP[1:0], are read-only bits.

Figure 4.9 Cause Register



BD **Branch Delay** **31**

When set, this bit indicates that the last exception was taken while the CW4011 was executing an instruction in a Branch Delay slot.

BT **Branch Taken** **30**

If the BD bit is set, this bit indicates if the branch was taken.

BT bit	Branch Condition
0	Branch Not Taken
1	Branch Taken

CE[1:0] **Coprocessor Error** **[29:28]**

The value in the coprocessor error field indicates the coprocessor unit referenced when a Coprocessor Unusable exception is taken:

CE1	CE0	Coprocessor Referenced
1	1	Coprocessor 3
1	0	Coprocessor 2
0	1	Coprocessor 1
0	0	Coprocessor 0

R **Reserved** **[27:16, 7, 1:0]**

These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

IP[7:0] **Interrupt Pending** **[15:8]**
This bit field indicates which interrupts are pending. Bits IP[7:2] correspond to the six external hardware interrupts and bits IP[1:0] correspond to the two software interrupts. The software interrupts can be set and cleared directly by writing to IP[1:0].

ExcCode[4:0] **Exception Code** **[6:2]**
This field defines the exception code. Table 4.3 lists the valid exception code values.

Table 4.3 Cause Register ExcCode Field

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	Bus	Bus error exception
7	—	Reserved
8	Sys	Syscall exception
9	Bp	Breakpoint exception
10	RI	Reserved instruction exception
11	CpU	Coprocessor Unusable exception
12	Ov	Arithmetic overflow exception
13	Tr	Trap exception
14	—	Reserved
15	FPE	Floating-point exception
16–31	—	Reserved

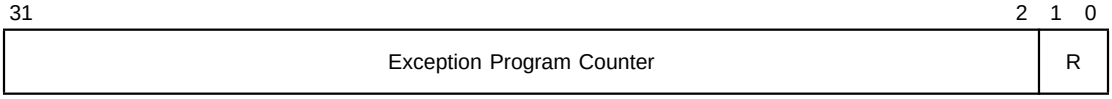
4.3.8 Exception Program Counter (EPC) Register

The read-write EPC register contains the address at 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
- 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)

Figure 4.10 shows the format of the EPC register. Bits [31:2] make up the program counter. Bits [1:0] are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

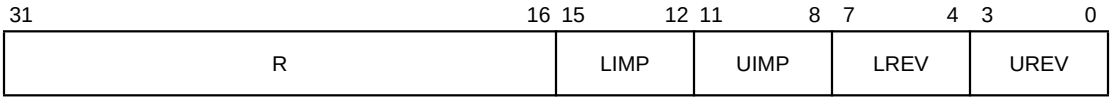
Figure 4.10 EPC Register



4.3.9 Processor Revision Identifier (PRId) Register

The 32-bit, read-only PRId register contains information identifying the implementation and revision level of the CW4011 core, as shown in Figure 4.11.

Figure 4.11 PRId Register



R **Reserved** **[31:16]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

LIMP **LSI Logic Implementation Number [15:12]**
This value represents the implementation number of the CW4011; it is currently set to 0x4.

UIMP **User Implementation Number [11:8]**
The value in this field represents the user's implementation number. This field can be programmed at the core interface using the iMPLop[3:0] lines.

LREV **LSI Logic Revision Number [7:4]**
This value is the revision number of the CW4011, which is set to 0x1 for the original version.

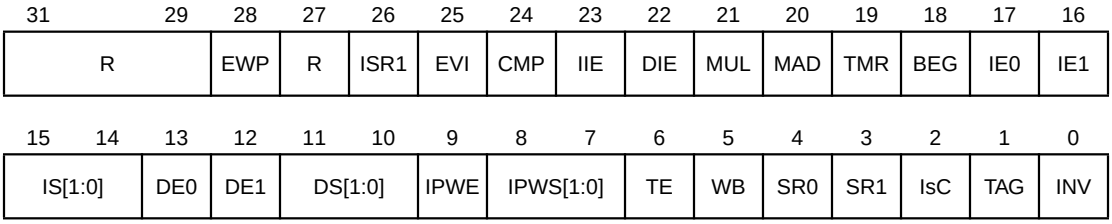
UREV **User Revision Number [3:0]**
The value of this field is interpreted as a processor unit revision number. This field can be programmed at the core interface using the REVLop[3:0] lines.

The revision number can distinguish between some chip revisions. However, LSI Logic does not guarantee that changes to the core will necessarily be reflected in the PRId register, or that changes to the revision number necessarily reflect real core 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 core.

4.3.10 Configuration and Cache Control (CCC) Register

The CCC register allows software to configure various pieces of the CW4011 design (for example, BIU, TLB, and cache controllers). Figure 4.12 shows the format of the CCC register.

Figure 4.12 CCC Register



R **Reserved** **[31:29, 27]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

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

ISR1 **I-Cache Scratchpad RAM** **26**
Setting this bit to one enables I-cache Set 1 to be used as a scratchpad RAM. Clearing ISR1 to zero disables the I-cache Set 1 scratchpad RAM mode.

EVI **External Vectored Interrupt** **25**
This bit enables and disables external vectored interrupts. Setting the bit to one enables the interrupt and clearing it to zero disables the interrupt.

CMP **R3000 Compatibility** **24**
This bit enables and disables the R3000 exception processing and status register compatibility mode. Setting the bit to one enables the mode and clearing it disables the mode.

IIE **I-Cache Invalidate Enable** **23**
This bit enables and disables I-cache invalidation. Setting IIE to one enables the interface and clearing it to zero disables the interface.

DIE **D-Cache Invalidate Enable** **22**
This bit enables and disables the D-cache invalidate interface. Setting the bit to one enables the request and clearing it to zero disables the interface.

MUL **Multiplier Enable** **21**
This bit enables and disables the hardware multiplier. Setting MUL to one enables the multiplier and clearing it disables the multiplier.

MAD **Multiplier Accumulate Extensions** **20**
This bit allows the multiplier to support accumulate extensions. Setting the bit to one enables the feature and clearing the bit disables the feature. When this bit is set, MUL must also be set.

TMR **Timer** **19**
This bit is the timer facility enable. When set to one, external hardware interrupt 5 is disabled. In the place of interrupt 5, the core enables the CP0 Count/Compare timer facility. This new timer facility replaces interrupt 5 in the Cause register IP[7] bit.

BEG **BIU Bus Enable Grant** **18**
This bit enables and disables the BIU bus grant. Setting this bit to one enables the external bus master. Clearing it to zero causes the CW4011 core to ignore the external bus master.

IE0 **I-Cache Set 0 Enable** **17**
This bit enables and disables Set 0 of the I-cache. Setting the bit to one enables Set 0 and clearing it to zero disables Set 0.

IE1 **I-Cache Set 1 Enable** **16**
This bit enables and disables Set 1 of the I-cache. Setting the bit to one enables Set 1 and clearing it to zero disables Set 1.

IS[1:0] **I-Cache Size** **[15:14]**

The IS[1:0] field determines the size of each I-cache set. The field is set as follows:

IS[1]	IS[0]	Cache Size (Kbytes)
0	0	1
0	1	2
1	0	4
1	1	8

DE0 **D-Cache Set 0 Enable** **13**

This bit enables and disables Set 0 of the D-cache. Setting the bit to one enables Set 0 and clearing it to zero disables Set 0.

DE1 **D-Cache Set 1 Enable** **12**

This bit enables and disables Set 1 of the D-cache. Setting the bit to one enables Set 1 and clearing it to zero disables Set 1.

DS[1:0] **D-Cache Size** **[11:10]**

The DS[1:0] field determines the size of each D-cache set. The field is set as follows:

DS[1]	DS[0]	Cache Size (Kbytes)
0	0	1
0	1	2
1	0	4
1	1	8

IPWE **In-Page Write Enable** **9**

This bit enables and disables in-page write operations. Setting the bit to one enables in-page write and clearing it to zero disables in-page write.

IPWS[1:0] **In-Page Write Size** **[8:7]**

The IPWS[1:0] field determines the external DRAM page size for in-page write operations. The field is set as follows:

IPWS[1]	IPWS0	In-Page Write Size (Kbytes)
0	0	1
0	1	2
1	0	4
1	1	8

TE **TLB Enable** **6**

This bit enables and disables the TLB. Setting the bit to one enables the TLB, if one is present, and clearing the bit to zero disables the TLB.

WB **WriteBack** **5**

This bit defines the caching algorithm used for *kseg0*. Additionally, when the TLB is absent or disabled, it also defines the caching algorithm for *kuseg* and *kseg2*. Setting WB to one enables WriteBack operation and clearing WB to zero enables WriteThrough operation.

SR0 **Scratchpad RAM Mode Set 0** **4**

This bit enables and disables scratchpad RAM mode for Set 0 of the D-cache. Setting the bit to one enables scratchpad mode and clearing it to zero disables scratchpad mode.

SR1 **Scratchpad RAM Mode Set 1** **3**

This bit enables and disables scratchpad RAM mode for Set 1 of the D-cache. Setting the bit to one enables scratchpad mode and clearing it to zero disables scratchpad mode.

IsC **Isolate Cache** **2**

This bit enables isolate cache mode. This means that stores to the cache are not propagated to external memory. Setting the bit to one enables the mode and clearing it to zero disables the mode.

TAG **Tag Test Mode** **1**

This bit enables and disables tag test mode, which is used for cache maintenance. Setting the bit to one

enables the mode and clearing it to zero disables the mode.

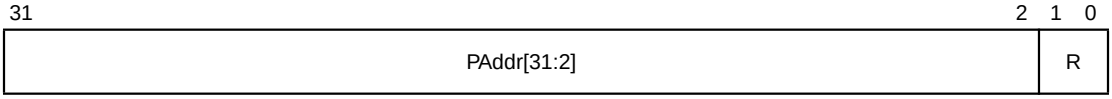
INV **Invalidate Cache Mode** **0**
This bit enables and disables cache invalidate mode, which is used for cache maintenance. Setting the bit to one enables the mode and clearing it to zero disables the mode.

4.3.11 Load Linked Address (LLAddr) Register

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

Figure 4.13 shows the format of the LLAddr register. Bits [31:2] contain the physical address (PAddr). Bits [1:0] are reserved and cleared to zero.

Figure 4.13 LLAddr Register

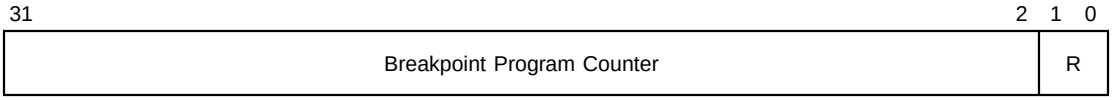


4.3.12 Breakpoint Program Counter (BPC) Register

Software uses the read/write BPC register to specify a program counter breakpoint. The BPC register is used in conjunction with the Breakpoint PC Mask register, described in Section 4.3.14.

Figure 4.14 shows the format of the 32-bit BPC register. Bits [1:0] are reserved and cleared to zero.

Figure 4.14 BPC Register

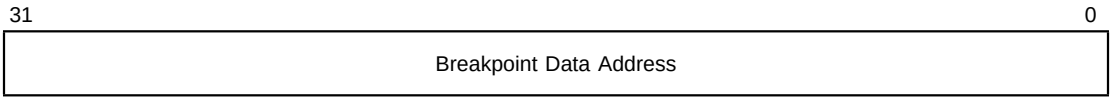


4.3.13 Breakpoint Data Address (BDA) Register

Software uses the read/write BDA register to specify a virtual data address breakpoint. The BDA register is used in conjunction with the Breakpoint Data Address Mask register described in Section 4.3.15.

Figure 4.15 shows the format of the 32-bit BDA register.

Figure 4.15 BDA Register

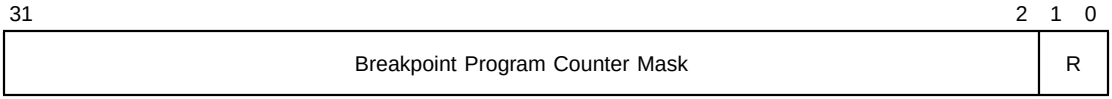


4.3.14 Breakpoint PC Mask (BPCM) Register

The read/write BPCM register masks bits in the BPC register. A one in any bit in the BPCM register indicates that the CW4011 compares the corresponding PC bit to that contained in the BPC register for program counter exceptions. Zero values in the mask indicate that the CW4011 does not check the corresponding PC bits to the BPC register bits.

Figure 4.16 shows the format of the 32-bit BPCM register. Bits [1:0] are reserved and cleared to zero.

Figure 4.16 BPCM Register

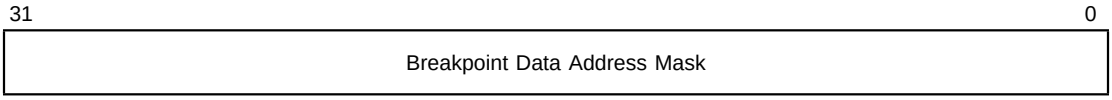


4.3.15 Breakpoint Data Address Mask (BDAM) Register

The read/write BDAM register masks bits in the BDA register. A one in any bit in the BDAM register indicates that the CW4011 compares the corresponding virtual data address bit to that contained in the BDA register for data address (debug) exceptions. Values of zero in the mask indicate that the CW4011 does not check the corresponding virtual data address bits to the BDA register bits.

Figure 4.17 shows the format of the 32-bit BDAM register.

Figure 4.17 BDAM Register



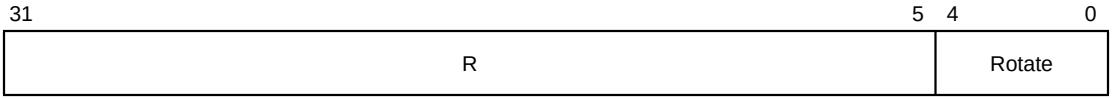
4.3.16 Rotate Register

Select and Rotate Left (SELSL) and Select and Rotate Right (SELSR) use the lower five bits of the Rotate register [4:0] as the shift count. This is useful for data alignment operations in graphics and in bit-field selection routines for data transmission and compression applications.

Even though the Rotate register resides in the CP0, User-mode access to the register is always granted, regardless of the value contained in the Cu0 bit of the Status register.

Figure 4.18 shows the format of the Rotate register.

Figure 4.18 Rotate Register



R	**Reserved** These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.	**[31:5]**
Rotate	**Rotate** This field determines the shift count.	**[4:0]**

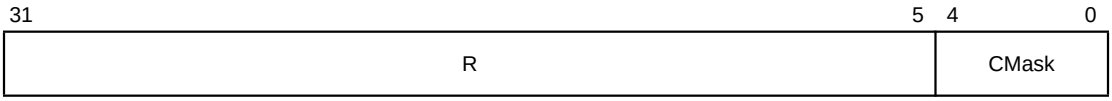
4.3.17 Circular Mask (CMask) Register

The CMask register is used by the CW4011 instruction set extensions. The Load/Store word/halfword/byte with update circular instructions store a value in the destination register and update the base address register with the addition of base + offset, which is modified according to the value of bits [4:0]. This feature is important in DSP (digital signal processing) and other applications that use circular buffers.

Even though the CMask register resides within the CP0, User-mode access to the register is always granted, regardless of the value contained in Status[Cu0].

Figure 4.19 shows the format of the CMask register.

Figure 4.19 CMask Register



R	**Reserved** These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.	**[31:5]**
CMask	**Circular Mask** This field contains the circular mask.	**[4:0]**

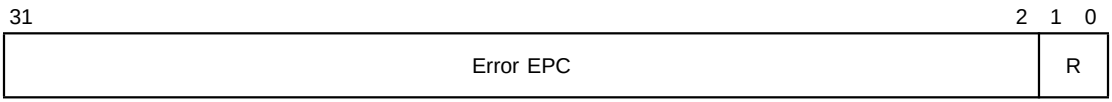
4.3.18 Error Exception Program Counter (Error EPC) Register

The Error EPC register is similar to the EPC register. It stores the PC (Program Counter) on cold reset, warm reset, and NMI exceptions. The read/write Error EPC register contains the virtual address at which instruction processing can resume after the exception has been serviced. The address may be either:

- The virtual address of the first instruction terminated by the exception
- The virtual address of the immediately preceding branch or jump instruction when the terminated instruction is in a Branch Delay slot

There is no Branch Delay slot indication for the Error EPC register. Figure 4.20 shows the format of the Error EPC register. Bits [31:2] make up the Error EPC. Bits [1:0] are reserved and cleared to zero.

Figure 4.20 Error EPC Register



4.4 Exception Description Details

This section describes each of the CW4011 core exceptions, what causes these exceptions, and how they are handled and serviced. This section is further divided as follows:

- Section 4.4.1, “Exception Operation”
- Section 4.4.2, “Precision of Exceptions”
- Section 4.4.3, “Exception Vector Locations”
- Section 4.4.4, “Priority of Exceptions”
- Section 4.4.5, “Reset Exceptions”
- Section 4.4.6, “Interrupt Exceptions”
- Section 4.4.7, “Address Error Exception”
- Section 4.4.8, “TLB Exceptions”
- Section 4.4.9, “Bus Error Exception”

- Section 4.4.10, “Integer Overflow Exception”
- Section 4.4.11, “Trap Exception”
- Section 4.4.12, “System Call Exception”
- Section 4.4.13, “Breakpoint Exception”
- Section 4.4.14, “Reserved Instruction Exception”
- Section 4.4.15, “Floating-Point Exception”
- Section 4.4.16, “Coprocessor Unusable Exception”
- Section 4.4.17, “Debug Exception”

4.4.1 Exception Operation

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

When an exception occurs, the EPC register is loaded with the restart location at which execution can resume after the exception has been serviced. The EPC register contains the address of the instruction associated with the exception, or, if the instruction was executing in a Branch Delay slot, the EPC register contains the address of the branch instruction immediately preceding.

4.4.1.1 R4000 Mode Operation (Default after Cold Reset)

The CW4011 processor uses the following mechanisms for saving and restoring the operating mode and interrupt status:

- A single interrupt enable bit (IE) located in the Status register
- A base operating mode (User, Kernel) located in the KSU field of the Status register
- An exception level (normal, exception) located in the EXL field of the Status register
- An error level (normal, error) located in the ERL field of the Status register

Interrupts are enabled by setting the IE bit to one and both levels (EXL, ERL) to normal.

Table 4.4 shows how the current processor operating mode is defined.

Table 4.4 Current Processor Mode

Current Mode	Status KSU[1:0]	Status EXL	Status ERL
User	10	0	0
Kernel	00	0	0
Kernel	xx	1	0
Kernel	xx	0	1

Exceptions set the exception level to exception (EXL = 1). The exception handler typically resets the exception level to normal (EXL = 0) after saving the appropriate state. It sets it back to exception while restoring that state. Returning from an exception (ERET instruction) resets the exception level to normal.

4.4.1.2 R3000 Mode Operation

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

When responding to an exception, the current mode bits (KUc/IEc) are saved into the previous mode bits (KUp/IEp); the previous mode bits are saved into the old mode bits (KUo/IEo); and the current mode bits (KUc/IEc) are both cleared to zero.

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

4.4.1.3 Exception Processing Diagrams

Figures 4.21–4.25 show the basic set of actions taken for each of the major CW4011 exception classes: Cold Reset, Warm Reset, Nonmaskable Interrupt (NMI), Common, Debug, and External Vectored Interrupt.

Figure 4.21 Cold Reset Exception

```
Random ← TLBENTRIES - 1
Wired ← 0
CCC ← 0^32
DCS ← 0^32
ErrorPC ← PC
SR ← 0^4 || SR[27:23] || 1 || 0 || 0 || SR[19:3] || 1 || SR[1:0]
PC ← 0xBFC0 0000
```

Figure 4.22 Warm Reset, NMI Exceptions

```
ErrorPC ← PC
if (CCC24 = 0) then
    SR ← SR[31:23] || 1 || 0 || 1 || SR[19:3] || 1 || SR[1:0]
else
    SR ← SR[31:23] || 1 || 0 || 1 || SR[19:6] || SR[3:0] || 0^2
endif
PC ← 0xBFC0 0000
```

Figure 4.23 Common Exceptions

```
Cause ← BD || BT || CE || 0^12 || Cause[15:8] || 0 || ExcCode || 0^2
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ← PC
endif
if (CCC24 = 0) then
    SR ← SR[31:2] || 1 || SR0
else
    SR ← SR[31:6] || SR[3:0] || 0^2
endif
if (SR22 = 1) then
    if (CCC24 = 0) then
         PC ← 0xBFC0 0200 + vector offset
    else
         PC ← 0xBFC0 0100 + vector offset
    endif
else
    PC ← 0x8000 0000 + vector offset
endif
```

Figure 4.24 Debug Exception

```
DCS ←DCS[31:6] || T || W || R || DA || PC || DB
Cause ←BD || BT || Cause[29:0]
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ←PC
endif
if (CCC24 = 0) then
    SR ←SR[31:2] || 1 || SR0
else
    SR ←SR[31:6] || SR[3:0] || 0^2
endif
if (SR22 = 1) then
    if (CCC24 = 0) then
         PC ←0xBFC0 0200 + vector offset
    else
         PC ←0xBFC0 0100 + vector offset
    endif
else
    PC ←0x8000 0000 + vector offset
endif
```

Figure 4.25 External Vectored Interrupt Exception

```
Cause ←BD || BT || Cause[29:0]
if ((CCC24 = 1) | (SR1 = 0)) then
    EPC ←PC
endif
if (CCC24 = 0) then
    SR ←SR[31:2] || 1 || SR0
else
    SR ←SR[31:6] || SR[3:0] || 0^2
endif
PC ←EXVAp[31:2] || 0^2
```

4.4.2 Precision of Exceptions

Exceptions are logically precise. This means that the instruction that causes an exception and all those that follow it are aborted, generally before committing to any state; execution picks up where it left off before the exception; and the instruction can be re-executed after the exception has been serviced. When following instructions are killed, exceptions associated with those instructions are also killed, so that exceptions are not taken in the order detected, but in the instruction fetch order.

Interrupts generated by external devices attached to the processor have a variety of meanings, depending on the system environment into which the CW4011 core is designed. Variations in memory system design can affect the meaning of bus error exceptions and the location and means of accessing relevant parameters to service them. As far as possible, this

architectural description of the exception handling system defines which state information is reliable and which is unreliable.

In some cases, however, the characteristics of the pipeline staging cannot guarantee that all states in the processor and associated system will remain completely unchanged as a result of the (possibly incomplete) execution of instructions immediately following an instruction that has caused an exception. State changes that may occur include the following:

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

These changes can normally be ignored because the state of the machine is sufficiently restored, allowing execution to resume after the exception has been serviced.

4.4.3 Exception Vector Locations

The Cold Reset, Warm Reset, and NMI exceptions are always vectored to location 0xBFC00000. Addresses for other exceptions are a combination of a vector offset and a base address, and they are determined by the BEV bit of the Status register. Table 4.5 shows the vector base addresses and Table 4.6 shows the vector offsets.

Table 4.5 Exception Vector Base Addresses

BEV	R4000 Mode (CCC24 = 0)	R3000 Mode (CCC24 = 1)
0	0x80000000	0x80000000
1	0xBFC00200	0xBFC0100

Table 4.6 Exception Vector Offset Addresses

Exception	R4000 Mode (CCC24 = 0)	R3000 Mode (CCC24 = 1)
TLB refill	0x000 (EXL = 0)	0x000 (*kuseg* access)
Debug	0x040	0x040
All Others	0x180	0x080

4.4.4 Priority of Exceptions

While more than one exception can occur for a single instruction, only one exception is reported. Table 4.7 shows the priority order given to the exception, with Cold Reset having the highest priority.

Table 4.7 Exception Priority Order

Priority	Exception
Highest	Cold Reset
	Warm Reset
	Nonmaskable Interrupt
	Address Error (Instruction Fetch)
	TLB Refill (Instruction Fetch)
	TLB Invalid (Instruction Fetch)
	Bus Error
	Integer Overflow, Trap, System Call, Breakpoint, Reserved Instruction, Coprocessor Unusable, Floating-Point Error
	Address Error (Data Access)
	TLB Refill (Data Access)
	TLB Invalid (Data Access)
	TLB Modified (Data Write)
	Interrupt
	External Vectored Interrupt
Lowest	Debug

4.4.5 Reset Exceptions

This subsection describes the cold and warm reset exceptions.

4.4.5.1 Cold Reset Exception

The primary purpose of a cold reset is to initialize the CW4011 core at power-up. This section describes the cause of and response to a Cold Reset exception.

Cause – The Cold Reset exception occurs when the CRESETn signal is asserted and then deasserted. This exception is not maskable.

Handling – The CPU provides a special interrupt vector (0xBFC00000) for the Cold Reset exception. The reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to handle the exception. The processor can fetch and execute instructions while the caches and virtual memory are in an undefined state.

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

- In the Status register, the CU[3:0] and SR bits are cleared to zero and the ERL and BEV bits are set to one. Other bits are undefined.
- The Random register is initialized to the value of its upper boundary.
- The Wired register is initialized to zero.

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

4.4.5.2 Warm Reset Exception

The primary purpose of the Warm Reset exception is to reinitialize the processor after a fatal error. Unlike nonmaskable interrupts, all cache and bus state machines are reset by this exception. Like Cold Reset, it can be used on the processor in any state. The caches, TLB, and normal exception vectors need not be properly initialized. This section describes the cause of and response to a Warm Reset exception.

Cause – The Warm Reset exception occurs when the WRESETn signal is asserted and then deasserted. This exception is not maskable.

Handling – The reset exception vector (0xBFC00000) is used for this exception. The vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache to handle the exception. The SR bit of the Status register is set to distinguish between a Warm Reset exception and a Cold Reset exception.

The contents of all registers are preserved when the Warm Reset exception occurs, except for the following:

- The ErrorPC register, which contains the restart PC (Program Counter)
- The BEV and SR bits of the Status register, which are set to one
- R4000 mode, in which the ERL bit is set to one
- R3000 mode, in which KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0

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

Servicing – The Warm Reset exception is serviced by saving the current processor state for diagnostic purposes, and reinitializing in a manner similar to that for the Cold Reset exception.

4.4.6 Interrupt Exceptions

This section describes exceptions caused by nonmaskable interrupts, normal interrupts, and external vectored interrupts.

4.4.6.1 Nonmaskable Interrupt (NMI) Exception

Nonmaskable interrupts cannot be disabled. They occur when a catastrophic event, such as power failure, requires immediate attention to maintain system integrity.

Cause – The Nonmaskable Interrupt exception occurs in response to the falling edge of the NMI signal. As the name implies, the NMI exception is not maskable, and occurs regardless of the settings of the EXL, ERL, and IE Status register bits.

Handling – The reset exception vector (0xBFC00000) is also used for this exception. The reset vector resides in unmapped and uncached CPU address space, so the hardware need not initialize the TLB or the cache

to handle the NMI interrupt. The SR bit of the Status register is set to differentiate the NMI exception from a Cold Reset exception.

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

Unlike Cold and Warm Reset, but in common with other exceptions, NMI is taken only at instruction boundaries. The states of the caches and memory system are preserved by this exception.

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

- The ErrorPC register, which contains the restart PC
- The BEV and SR bits of the Status register, which are set to one
- R4000 mode, in which the ERL bit is set to one
- R3000 mode, in which KUo/IEo ← KUp/IEp ← KUc/IEc ← 0/0

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

4.4.6.2 Interrupt Exception

This section describes the cause of and response to an Interrupt exception.

Cause – The Interrupt exception occurs when one of the eight interrupt conditions is asserted. The significance of these interrupts depends on the specific system implementation. Each of the eight interrupts can be masked by clearing the corresponding bit in the Interrupt Mask field of the Status register. All eight interrupts can be masked at once by clearing the IE bit of the Status register.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to INT.

The IP field of the Cause register indicates the current interrupt requests. It is possible that more than one of the bits will be set at the same time, or that no bits will be set if an interrupt is asserted and then deasserted before the Cause register is read.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

Servicing – If the interrupt is caused by one of the two software generated exceptions, the interrupt condition is cleared by setting the corresponding Cause register bit to zero.

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

4.4.6.3 External Vectored Interrupt Exception

The CW4011 implements an external vectored interrupt interface, which consists of an interrupt input (EXViNTn), interrupt vector virtual address input (EXVAp[31:2]), and interrupt accepted output (EXVAEn). The signals must be asserted and deasserted on the rising edge of the system clock. This interrupt class can be enabled or disabled using the EVI bit in the CCC register (enabled when CCC24 = 1). This section describes the cause of and response to an external vectored interrupt exception.

Cause – An external vectored interrupt occurs when the EXViNTn is asserted. The significance of this interrupt depends on the specific system implementation. The interrupt can be masked by clearing the IE (R3000 = IEc) bit of the Status register.

Handling – The virtual address specified by the EXVAp[31:2] interface is used to specify the target exception handling routine. The EXVAp[31:2] address must be provided by a user-defined interrupt controller. The EXViNTn and EXVAp[31:2] inputs must be held stable and valid until the exception is accepted. This is indicated by asserting the EXVAEn output for one cycle.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set as an indicator.

Servicing – The interrupt condition can be cleared in the user-defined interrupt controller in one of two ways: by detecting the assertion of the interrupt accepted output (EXVAEn), or by correcting the condition causing the interrupt pin (EXViNTn) to be asserted.

4.4.7 Address Error Exception

This section describes the cause of and response to an Address Error exception.

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

- Load, fetch, or store a word that is not aligned on a word boundary
- Load or store a halfword that is not aligned on a halfword boundary
- Reference the Kernel address space from User mode

The Address Error exception is not maskable.

Handling – The common exception vector is used for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: AdEL for a data load or instruction fetch, AdES for a data store operation.

When the Address Error exception occurs, the BadVAddr register retains the virtual address that was not properly aligned or that 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 points at the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The process executing at the time should be handed a “segmentation violation” signal. This error is usually fatal to the process incurring the exception.

4.4.8 TLB Exceptions

This subsection describes the TLB modified exception, TLB invalid exception, and the TLB refill exception. If a specific design does not have a TLB, this section may be disregarded.

4.4.8.1 TLB Refill Exception

This section describes the cause of and response to a TLB Refill exception.

Cause – The TLB Refill exception occurs when there is no TLB entry to match a reference to a mapped address space. This exception is not maskable.

Handling – A special TLB Refill exception vector is used for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: TLBL for a data load or instruction fetch and TLBS for a data store operation.

When the TLB refill exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed translation. The EntryHi register also contains the address space identifier (ASID) from which the translation fault occurred. The Random register normally contains a valid location in which to place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

R4000 mode – This special exception vector is used when the exception level (at the time of TLB miss detection) is set to normal (EXL = zero). If the exception level is set to exception (EXL = 1), the common exception vector is used.

R3000 mode – This special exception vector is used when User or Kernel mode references to user memory space (*kuseg*) do not find a matching entry in the TLB. If the reference is to kernel memory space (*kseg2*), the common exception vector is used.

Servicing – To service this exception, the contents of the Context register are used as a virtual address to fetch memory locations containing the physical page frame and access control bits for a TLB entry. This information is placed in the EntryHi and EntryLo registers and written into the TLB.

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. In this case, a TLB refill exception is allowed inside the TLB Refill handler. While the first exception goes to a special exception vector offset (0x000), the second exception goes to the common exception vector offset (0x180).

The second TLB refill exception obscures the contents of the BadVAddr, Context, and EntryHi registers within the TLB Refill handler. As a result, the exact virtual address whose translation caused the first fault is not known unless the TLB Refill handler specifically saved this address. It is possible to observe only the failing PTE virtual address. The BadVAddr register now contains the original contents of the Context register within the TLB Refill handler, which is the PTE address for the original failing address.

The operating system can determine the original virtual page number that caused the fault, but not the complete address. The operating system uses this information to fetch the PTE that contains the physical address and to access control information. It also writes the entry into the TLB and returns to the original user program.

Returning to the TLB Refill handler at this point should be avoided.

R4000 mode – When the EXL bit is set, it prevents the EPC from the first TLB refill exception from being overwritten by the second TLB refill exception. Consequently, the appropriate return address can be determined from the values of the current EPC and the BD bit of the Status register.

R3000 mode – The TLB Refill handler must save the first refill EPC and Status[BD] information in a way that allows the second refill to find it. Using this “saved” EPC register and Status[BD] information, the appropriate return address can be determined.

4.4.8.2 TLB Invalid Exception

This section describes the cause of and response to a TLB Invalid exception.

Cause – The TLB Invalid exception occurs when a virtual address reference matches a TLB entry that is marked invalid. This exception is not maskable.

Handling – The common exception vector is used for this exception. The Cause register ExcCode is set based on the type of reference that caused the exception: TLBL for a data load or instruction fetch, TLBS for a data store operation.

When the TLB Invalid exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed 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 place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception, unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The valid bit of the TLB entry is typically cleared when:

- 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)

After servicing the cause of this exception, the TLB entry is located with the TLB Probe (TLBP) instruction, and replaced by an entry with the valid bit set.

4.4.8.3 TLB Modified Exception

This section describes the cause of and response to a TLB Modified exception.

Cause – The TLB Modified exception occurs during a store operation, when the virtual address reference to memory matches a TLB entry that is marked valid but is not dirty or writable. This exception is not maskable.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to one, indicating a TLB modification exception (Mod).

When the TLB Modified exception occurs, the BadVAddr, Context, and EntryHi registers hold the virtual address that failed 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 place the replacement TLB entry. The contents of the EntryLo register are undefined.

The EPC register points at the instruction that caused the exception, unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

Servicing – The Kernel uses the failed virtual address and virtual page number to identify the corresponding access control information. The page identified may or may not permit write access. If writes are not permitted, a Write Protection Violation has occurred.

If write access is permitted, the Kernel marks the page frame as dirty/writable in the Kernel's own data structures. The TLBP instruction is used to place the index of the TLB entry that must be altered in the Index register. The EntryLo registers are loaded with physical page frame and access control bits (with the D bit set), and the EntryHi and EntryLo registers are written into the TLB.

4.4.9 Bus Error Exception

This section describes the cause of and response to a Bus Error exception.

Cause – The Bus Error exception occurs when signaled by board-level circuitry for events such as bus time-out, bus parity errors, and invalid physical memory accesses. This exception is not maskable.

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

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to Bus.

The EPC register points at the first instruction for which processing was not completed, unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

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

4.4.10 Integer Overflow Exception

This section describes the cause of and response to an Integer Overflow exception.

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

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to OV.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The process executing at the time of the exception should be handed an “integer overflow” signal. This error is usually fatal to the current process.

4.4.11 Trap Exception

This section describes the cause of and response to a Trap exception.

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

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to TR.

The EPC register points at the instruction that caused the exception unless the instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The process executing at the time of the exception should be handed a “trap” signal. This error is usually fatal.

4.4.12 System Call Exception

This section describes the cause of and response to a System Call exception.

Cause – The System Call exception occurs when an attempt is made to execute the SYSCALL instruction. This exception is not maskable.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to Sys.

The EPC register points at the SYSCALL instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in the Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

Servicing – When this exception occurs, control is transferred to the applicable system routine. To resume execution, the routine must restart

instruction execution after the SYSCALL instruction. This restart address can be computed using the EPC register along with the BD and BT bits in the Cause register.

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

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

4.4.13 Breakpoint Exception

This section describes the cause of and response to a Breakpoint exception.

Cause – The Breakpoint exception occurs when an attempt is made to execute the BREAK instruction. This exception is not maskable.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to BP.

The EPC register points at the BREAK instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set as an indicator.

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

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

- If (BD = 0) then Restart_PC = EPC + 4

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

When the BREAK instruction resides in a Branch Delay slot, it is up to the exception handler to obtain the Branch Target Address from the prior branch.

4.4.14 Reserved Instruction Exception

This section describes the cause of and response to a Reserved Instruction exception.

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

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to RI.

The EPC register points at the BREAK instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The Reserved Instruction exception can be used to trap to emulation routines for instructions not supported in the CW4011 instruction set. Once emulation has been completed, execution can be resumed using the EPC register along with the BD and BT bits in the Cause register.

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

When the instruction receiving a Reserved Instruction exception resides in a Branch Delay slot, it is up to the exception handler to obtain the Branch Target Address from the prior branch.

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

4.4.15 Floating-Point Exception

This section describes the cause of and response to a floating-point exception.

Cause – The Floating-Point exception is used by the floating-point coprocessor (if installed). The contents of the Floating-Point Control Status register (inside CP1) indicate the cause of the exception.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to FPE.

The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – This exception is cleared by clearing the appropriate bit in the Floating-Point Control Status register. For an unimplemented instruction exception, the Kernel should emulate the instruction. For other exceptions, the Kernel should pass the exception to the user process that caused the exception.

4.4.16 Coprocessor Unusable Exception

This section describes the cause of and response to a 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 for CP0 instructions, when the unit has not been marked usable and the process is executing in User mode.

This exception is not maskable.

Handling – The common exception vector is used for this exception. The ExcCode field in the Cause register is set to CPU. The contents of the

CE field in the Cause register indicate the coprocessor to which an attempted reference has been made.

The EPC register points at the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The coprocessor unit to which an attempted reference was made is identified by the CE field of the Cause register. The result is one of the following:

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

4.4.17 Debug Exception

This section describes the cause of and response to a Debug exception.

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

R4000 mode – In R4000 mode, the debug exception can be masked by setting the EXL bit in the Status register. When this bit is set, a debug event does not cause an exception trap even if the DCS[TE] bit is set to one. However, the status bits of the DCS register are updated to indicate that an event was recognized.

R3000 mode – In R3000 mode, the debug exception is not maskable.

Handling – The Debug exception vector is used for this exception in both R4000 and R3000 modes.

The EPC register points at the instruction that caused the exception unless this instruction is in a Branch Delay slot. If the instruction is in a Branch Delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Servicing – The Debug exception is a debugging aid. Typically the exception handler transfers control to a debugger, allowing you to examine the situation. The debug exception condition must be disabled to execute the failing instruction and then re-enabled.

Notes:

1. The Trace status bit (DCS5) is set whenever a branch instruction is encountered regardless of whether the branch is actually taken. However, if the debug exception trap is enabled (DCS31 = 1), an exception is recognized only if the branch is taken and the target instruction executed.
2. The Program Counter debug status bit (DCS1) is set whenever the target address of a branch falls within the specified PC address range (BPC, BPCM) regardless of whether the branch is actually taken. However, if the debug exception trap is enabled (DCS31 = 1), an exception is recognized only if the branch is taken and the target instruction executed.

Chapter 5
CW4011 Memory Management

This chapter describes the System Coprocessor (Coprocessor 0) Memory Management functions. It contains the following sections:

- Section 5.1, “TLB Physical Organization”
- Section 5.2, “Memory Management System”
- Section 5.3, “Virtual Memory and the TLB”

Please note that the translation lookaside buffer (TLB) is an optional module for the CW4011. If a specific design does not contain a TLB, any TLB references in this chapter may be ignored.

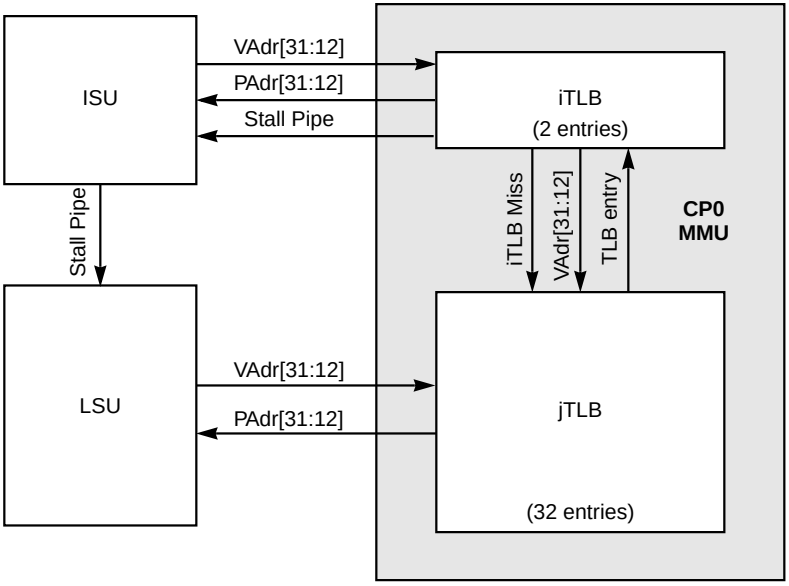
5.1 TLB Physical Organization

The physical implementation of the TLB consists of two main parts:

1. A two-entry instruction TLB (iTLB)
2. A 32-entry joint TLB (jTLB) that holds both instruction fetch and data access page translations

The CP0 can receive virtual address translation requests from both the ISU (instruction fetch) and the LSU (operand data access) during the same cycle. For maximum performance, address translations must occur in parallel. The two-piece TLB structure shown in Figure 5.1 addresses this problem by creating a separate two-entry TLB to be used for instruction fetch translations. With this structure, ISU and LSU fetches can be independently processed.

Figure 5.1 TLB Block Diagram



VAdr = Virtual address
PAdr = Physical address

The iTLB holds the two most recently used instruction fetch page translations. If a valid translation cannot be found in the iTLB, the CP0 must stall the pipeline for two cycles and search the jTLB for a valid entry. If the CP0 finds a valid entry in the jTLB, it copies it into the less recently used iTLB entry and processing continues. If a valid entry cannot be found, a TLB exception must be posted (see Chapter 4, “CW4011 Exception Processing,” for details.)

The entries in the iTLB are purged when the EntryHi register is written (for example, during a task switch). Consequently, the iTLB does not need to keep an eight-bit ASID for each entry. This reduces storage and match circuitry. This simplification should cause little or no performance penalty, because the entries probably need to be replaced anyway.

When no TLB is present in the system, the TE field of the Configuration and Cache Control (CCC) register is cleared to zero. This is transparent to the other modules in the CW4011 core. The CP0 modifies its translation behavior in the following manner:

- Physical Address[31:12] = Virtual Address[31:12]. For *kseg0* and *kseg1*, Physical Address [31:29] = 0; the same is true with TLB present.
- The caching algorithm used for each access is based on the address segment being accessed (*kuseg*, *kseg0*, and *kseg2* = cached; *kseg1* = uncached), and the CCC register fields (IE0, IE1, DE0, DE1, and WB). Table 5.1 shows the algorithm criteria for the I-cache and Table 5.2 lists criteria for the D-cache.

Table 5.1 I-Cache Algorithm Criteria

Address Segment	I-Cache Enabled	Ifetch Cache Algorithm
kuseg, *kseg0*, or *kseg2*	0	Uncached
	1	Cached
kseg1	X	Uncached

Table 5.2 D-Cache Algorithm Criteria

Address Segment	D-Cache Enabled	WB	D-Cache Algorithm
kuseg, *kseg0*, or *kseg2*	0	X	Uncached
	1	0	Cached, WriteThrough
	1	1	Cached, WriteBack
kseg1	X	X	Uncached

5.2 Memory Management System

The memory model used for the CW4011 processor is based on the R3000. To extend the CPU's address space, the virtual memory translates addresses composed in a large virtual address space into the physical memory system.

The CW4011 physical address space is four Gbytes and uses a 32-bit address. The virtual address is also 32 bits wide, and the maximum user process size is two Gbytes (2^{31}).

The virtual address is extended with an ASID to reduce the frequency of the TLB flushing when switching context. The size of the ASID is 8 bits. The ASID is contained in the CP0 EntryHi register and is described in the subsection entitled "EntryHi Register" on page 5-10.

5.2.1 Operating Modes

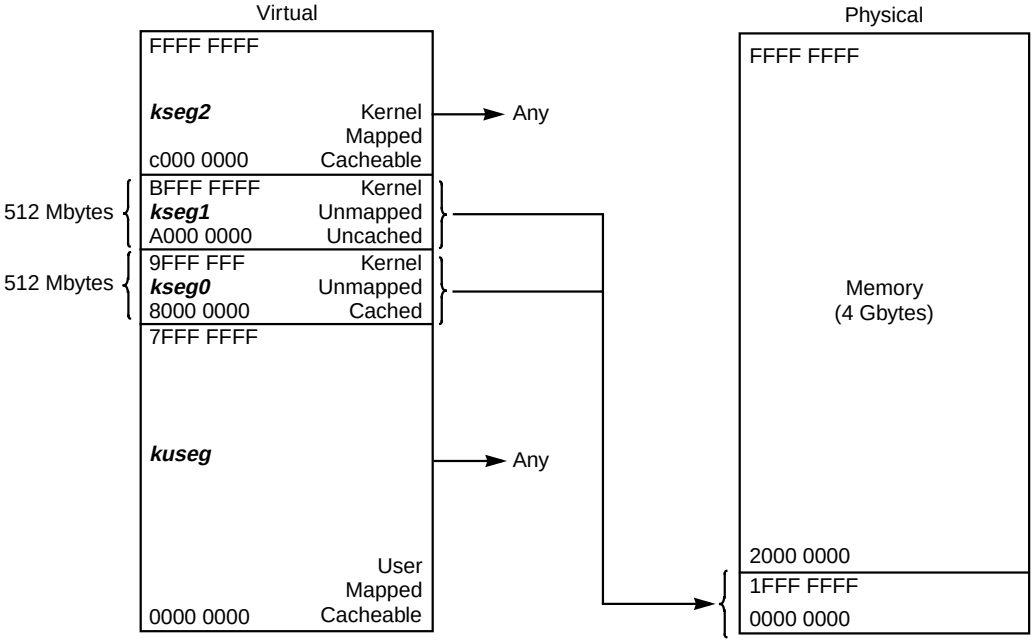
This section describes the two modes for 32-bit CW4011 operation:

- User mode, where nonsupervisory programs are executed
- Kernel mode, which is analogous to the "supervisory" mode provided by many machines

The CW4011 usually operates in User mode until an exception forces it into Kernel mode. It remains in Kernel mode until a Restore From Exception instruction (R3000 mode), or Exception Return (R4000 mode) instruction is executed to restore the processor to the mode existing prior to the exception.

Address mapping is different for Kernel and User modes. To simplify the management of user state from within the Kernel, the user-mode address space is a subset of the Kernel-mode address space. Figure 5.2 shows the virtual-to-physical memory map for both the User mode and Kernel mode segments.

Figure 5.2 CW4011 Virtual Memory Map



5.2.2 User Mode Virtual Addressing

In User mode, a single, uniform virtual address space (*kuseg*) of two Gbytes (2^{31} bytes) is available. The User segment starts at address 0x00000000, and all valid accesses have the most-significant bit cleared to zero. Referencing an address with the most significant bit set while in User mode causes an Address Error exception. The TLB maps all references to *kuseg* identically for either mode, and controls cache accessibility. *Kuseg* is typically used to hold user code and data, as well as the current user process. The processor state definition of User and Kernel modes description can be found in Section 4.3.6, “Status Register.”

5.2.3 Kernel Mode Virtual Addressing

As shown in Figure 5.2, the virtual address space is divided into regions, differentiated by the high-order bits of the address:

kuseg Starts at virtual address 0x00000000 and is 2 Gbytes long. It allows selective caching and mapping on a per-page basis, rather than requiring an all or nothing approach. This segment overlaps Kernel memory accesses with User memory accesses as described previously.

kseg0 Starts at virtual address 0x80000000 and is 512 Mbytes long. CW4011 direct maps references within *kseg0* onto the first 512 Mbytes of physical memory. These references use cache memory, but do not use the TLB for address translation. Thus, *kseg0* is typically used for kernel executable code and some kernel data.

kseg1 Starts at virtual address 0xA0000000 and is 512 Mbytes long. CW4011 direct maps references within *kseg1* onto the first 512 Mbytes of physical memory. These references do not use cache memory or the TLB for address translation. Thus, *kseg1* is typically used by operating systems for I/O registers, ROM code and disk buffers.

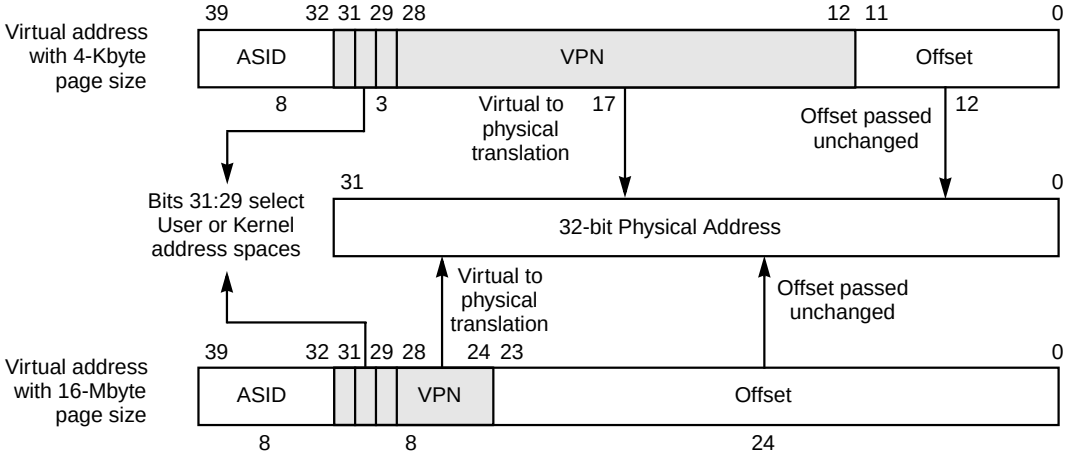
kseg2 Starts at virtual address 0xC0000000 and is 1024 Mbytes long. Like *kuseg*, it uses TLB entries to map virtual addresses to arbitrary physical ones, with or without caching. An operating system typically uses *kseg2* for stacks and per-process data that must remap on context switches. The operating system also uses *kseg2* for user page tables and some dynamically allocated data areas.

5.3 Virtual Memory and the TLB

Mapped virtual addresses are translated into physical addresses using an on-chip TLB. The TLB is a fully-associative memory that holds 32 entries that provide mapping to 32 physical page frames. The address range mapped by a page can be either 4 Kbytes or 16 Mbytes in size. When address mapping is indicated, each TLB entry is simultaneously matched against the virtual address extended by the current ASID stored in the EntryHi register.

If there is a match (hit), the physical page number is extracted from the TLB and concatenated with the offset to form the physical address, as shown in Figure 5.3.

Figure 5.3 CW4011 Virtual Address Format



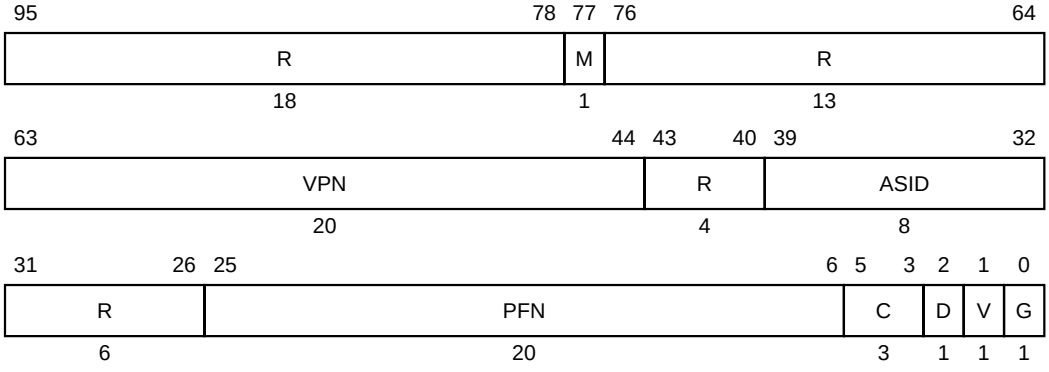
If no match occurs (a page miss), an exception is taken. Typically, software refills the TLB from a page table maintained by the system. Software can write over a selected TLB entry or use a hardware mechanism to write into a random location.

The CW4011 does not support the TLB-shutdown (TS) bit in the Status register, which indicates that more than one entry in the TLB matches the virtual address being translated. If more than one TLB entry matches the virtual address, the virtual address may be translated to an incorrect physical address. *System software must ensure that this situation is never created*.

5.3.1 TLB Entry Format

Figure 5.4 shows the 32-bit addressing TLB entry format the CW4011 uses. Each field of an entry has a corresponding field in the EntryHi, EntryLo, or PageMask registers described in sections 5.3.2.1, 5.3.2.2, and 5.3.2.3.

Figure 5.4 Format of CW4011 TLB Entry



R **Reserved** **[95:78, 76:64, 43:40, 31:26]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

M **Mask** **77**
This bit is the Page Mask bit. It is set to one for a 16-Mbyte page and cleared to zero for a 4-Kbyte page.

VPN **Virtual Page Number** **[63:44]**
This field contains the Virtual Page Number.

ASID **Address Space ID Field** **[39:32]**
This field contains the Address Space ID.

PFN **Page Frame Number** **[25:6]**
This field contains the Page Frame Number. This is the upper bits of the physical address.

C **Cache** **[5:3]**
This field contains the Cache algorithm, which specifies whether references to the page should be cached. If the references are to be cached, you can select one of two

algorithms: WriteBack or WriteThrough. The following table shows how the Cache bits are decoded.

CBit Settings			Value	Algorithm
0	0	0	0	Reserved
0	0	1	1	Reserved
0	1	0	2	Uncached
0	1	1	3	Cacheable—WriteThrough
1	0	0	4	Reserved
1	0	1	5	Reserved
1	1	0	6	Reserved
1	1	1	7	Cacheable—WriteBack

D **Dirty** **2**
If this bit is set to one, it indicates that the page marked is dirty and writable.

V **Valid** **1**
If this bit is set to one, it indicates that the TLB entry is valid.

G **Global** **0**
If this bit is set to one, the contents of the ASID field are ignored during TLB lookup.

5.3.2 TLB Support Registers

Table 5.3 lists the TLB registers used in association with the CP0 TLB.

Table 5.3 TLB Support Registers

Name	CP0 Register Number	Reference Page
EntryHi Register	10	5-10
EntryLo Register	2	5-11
PageMask Register	5	5-12
Index Register	0	5-13
Random Register	1	5-13
Wired Register	6	5-14

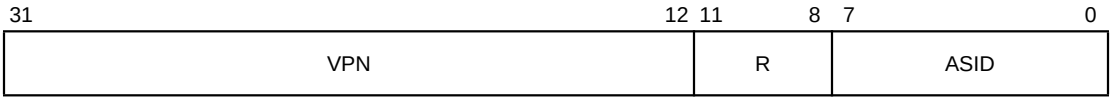
5.3.2.1 EntryHi Register

The EntryHi register is a read/write register used to access the TLB. In addition, this register contains the current ASID value for the processor. The ASID value is used to match the virtual address with a TLB entry during virtual address translation. Typically, the operating system assigns a unique ASID value to each known process. In this way, mappings held in the TLB are made unique to the process whose ASID they match.

The EntryHi register holds the high-order bits of a TLB entry when performing TLB read and write operations. When either a TLB refill, TLB invalid, or TLB modified exception occurs, the EntryHi register is loaded with the Virtual Page Number (VPN) and the ASID of the virtual address that failed to have a matching TLB entry.

EntryHi is accessed by the TLBP, TLBW, TLBWI, and TLBR instructions. Figure 5.5 shows the format of this register.

Figure 5.5 EntryHi Register



VPN **Virtual Page Number** **[31:12]**
This field contains the Virtual Page Number.

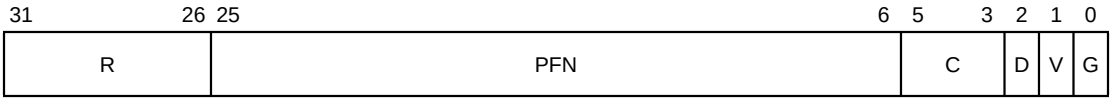
R **Reserved** **[11:8]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

ASID **Address Space ID** **[7:0]**
This field contains the Address Space ID.

5.3.2.2 EntryLo Register

The EntryLo register is a read/write register used to access the TLB. When performing read and write operations, the register contains a physical page frame number, cache algorithm, page dirty, translation valid, and global entry information. Figure 5.6 shows the format of this register.

Figure 5.6 EntryLo Register



R **Reserved** **[31:26]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

PFN **Physical Page Frame Number** **[25:6]**
This field contains the Physical Page Frame Number.

C **Cache** **[5:3]**
This field contains the Cache algorithm, which specifies whether references to the page should be cached. If the references are to be cached, you can select one of two algorithms: WriteBack or WriteThrough. The following table shows how the Cache bits are decoded.

CBit Settings			Value	Algorithm
0	0	0	0	Reserved
0	0	1	1	Reserved
0	1	0	2	Uncached
0	1	1	3	Cacheable—WriteThrough
1	0	0	4	Reserved
1	0	1	5	Reserved
1	1	0	6	Reserved
1	1	1	7	Cacheable—WriteBack

D **Dirty** **2**
If this bit is set to one, it indicates that the marked page is dirty and writable.

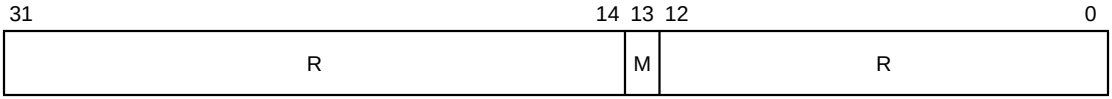
V **Valid** **1**
If this bit is set to one, it indicates that the TLB entry is valid.

G **Global** **0**
If this bit is set to one, the contents of the ASID field are ignored during TLB lookup. Mapping is globally available to all ASIDs.

5.3.2.3 PageMask Register

The PageMask register is a read/write register used to access the TLB. It implements a variable page size by holding a per-entry comparison mask. When virtual addresses are presented for translation, the corresponding PageMask bit in the TLB specifies whether or not virtual address bits [23:12] participate in the comparison. Figure 5.7 shows the format of the PageMask register.

Figure 5.7 PageMask Register



R **Reserved** **[31:14, 12:0]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

M **Mask** **13**
This field contains the PageMask. The following table shows the page size and the physical and virtual address bits for each setting of the Mask bit.

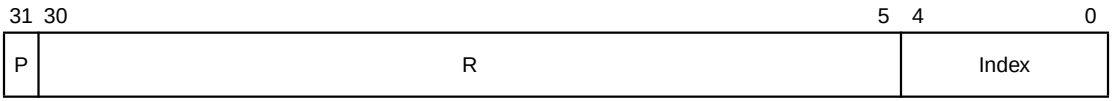
Mask Bit	Page Size	Physical Address	Virtual Address
1	16 Mbytes	PFN[31:24]	[23:0]
0	4 Kbytes	PFN[31:12]	[11:0]

5.3.2.4 Index Register

The Index register is a 32-bit, read/write register containing five bits that are used to index an entry in the TLB. The high-order bit indicates the success or failure of a TLB Probe (TLBP) instruction.

The Index register also specifies the TLB entry that is affected by the TLB Read (TLBR) and TLB Write Index (TLBWI) instructions. Figure 5.8 shows the format of the Index register.

Figure 5.8 Index Register



P **Probe** **31**
If this bit is set to one, it indicates that the last TLBP instruction failed to find a match.

R **Reserved** **[30:5]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

Index **Index** **[4:0]**
This field contains the index to the TLB entry. The TLBR and TLBWI instructions use this index.

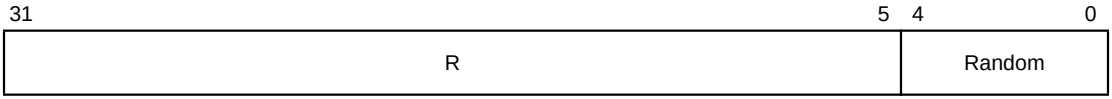
5.3.2.5 Random Register

The Random register is a 32-bit read-only register that contains five bits that are used to index an entry in the TLB. The register decrements for each clock cycle. The values range between a lower bound set by the number of TLB entries reserved for exclusive use by the operating system (defined in the Wired register), and an upper bound set by the total number of TLB entries (32 maximum).

The Random register specifies the entry in the TLB affected by the TLB Write Random (TLBWR) instruction. The register does not need to be read for this purpose, but the register can be read to verify proper operation.

To simplify testing, the Random Register is set to the value of the upper bound when the system is reset. It is also set to its upper bound when the Wired register is written. The format of this register is shown in Figure 5.9.

Figure 5.9 Random Register



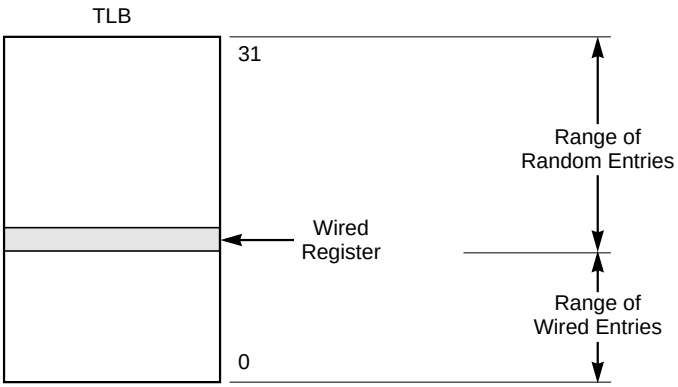
R **Reserved** **[31:5]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

Random **Random** **[4:0]**
This field contains the index to the TLB entry affected by the TLBWR instruction.

5.3.2.6 Wired Register

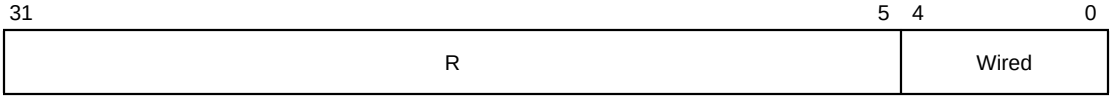
The Wired register is a read/write register that specifies the boundary between the wired (fixed, nonreplaceable entries that cannot be over-written by a TLBWR operation) and random entries of the TLB. Figure 5.10 shows the location in the TLB of the wired register.

Figure 5.10 Wired Register Location



When the system is reset, the Wired register is set to zero. Writing the register also sets the Random register to the value of its upper bound. Figure 5.11 shows the format of the Wired register.

Figure 5.11 Wired Register



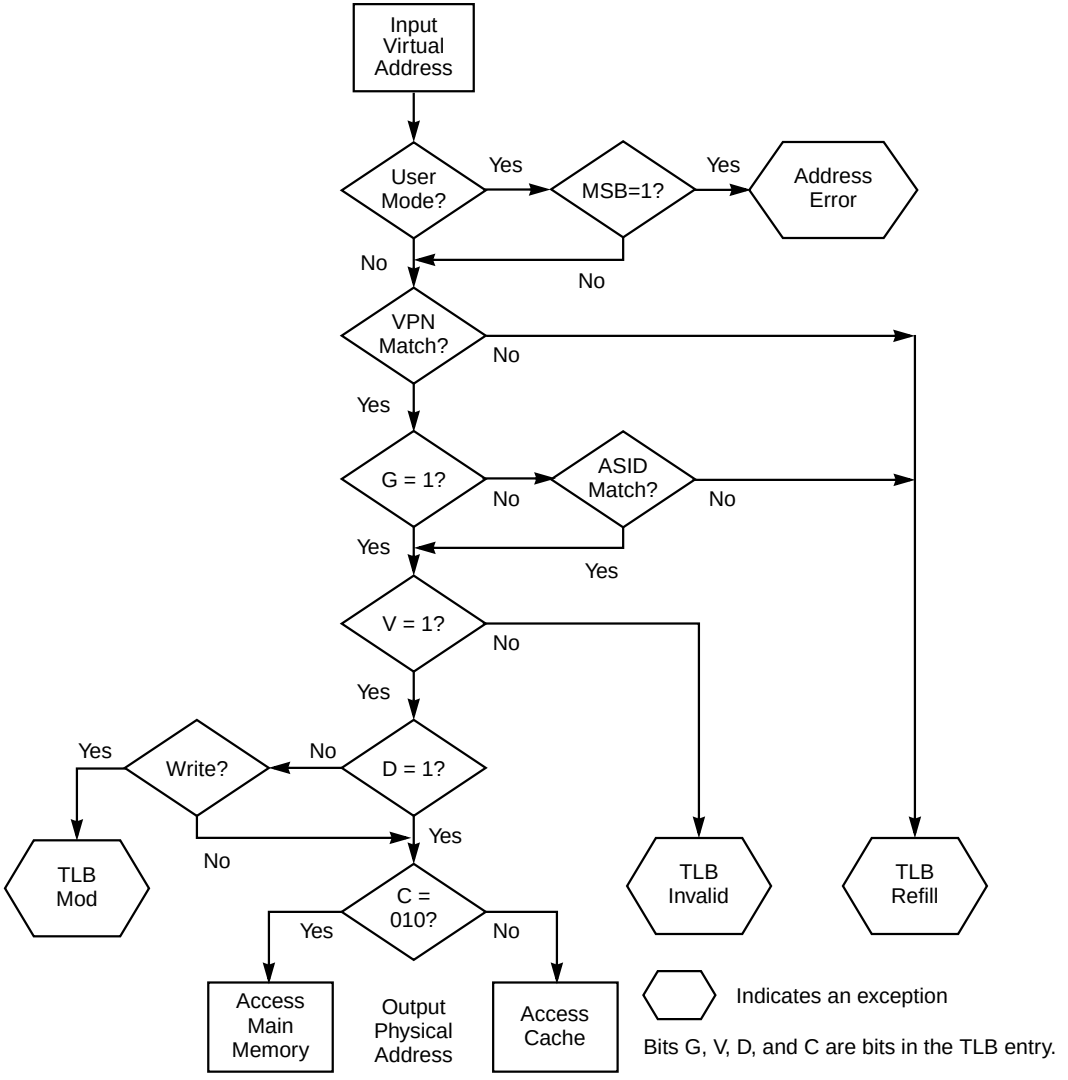
R **Reserved** **[31:5]**
These bits are not used and are read as zero. The CW4011 ignores attempts to set these bits; however, software should write these bits as zero to ensure compatibility with future versions of the software.

Wired **Wired** **[4:0]**
This field defines the lower boundary of Random TLB entries.

5.3.3 Virtual Address Translation

During virtual-to-physical address translation, the CP0 compares the ASID and the highest 7 to 20 bits of the virtual address to the contents of the TLB. The virtual address bits compared with the ASID depend on the page size. Figure 5.12 illustrates the TLB address translation process.

Figure 5.12 CW4011 TLB Address Translation Process



A virtual address matches a TLB entry under one of two conditions:

- The VPN field of the virtual address equals the VPN field of the entry and the G bit of the TLB entry is set.
- The ASID held in the EntryHi register matches the ASID field in the TLB entry.

Although the V bit of the TLB entry must be set for a valid translation to take place, it is not involved in determining a matching TLB entry.

If a TLB entry matches, the physical address and access control bits (C, D, and V) are retrieved from the entry. If no match is found, a TLB miss exception occurs. If the access control bits (D and V) indicate that the access is not valid, a TLB modification or TLB invalid exception occurs, respectively. If the C bits equal 0b010, the physical address that is retrieved is used to access main memory, bypassing the cache.

5.3.4 TLB Instructions

Table 5.4 lists the instructions that the CW4011 provides for working with the TLB.

Table 5.4 TLB Instruction

Instruction	Description
TLB Probe (TLBP)	The Index register is loaded with the address of the TLB entry whose contents match the contents of the EntryHi register. If no TLB entry matches, the highest order bit of the Index register is set. Results are undefined if a TLB reference encounters more than one matching TLB entry.
TLB Read (TLBR)	This instruction loads the EntryHi, EntryLo, and PageMask registers with the contents of the TLB entry specified by the Index register.
TLB Write Index (TLBWI)	This instruction loads the TLB entry specified by the Index register with the contents of the EntryHi, EntryLo, and PageMask registers.
TLB Write Random (TLBWR)	This instruction loads the TLB entry specified by the Random register with the contents of the EntryHi, EntryLo, and PageMask registers.

Notes:

1. If the TLB is not present or not enabled in the system, the CP0 reflects a Coprocessor Unusable exception if an attempt is made to execute any of the TLB instructions.
2. TLB instructions (TLBP, TLBR, TLBWI, and TLBWR) cannot be immediately preceded or followed by a data load instruction that requires target address translation (that is, *kuseg* and *kseg2*).

3. The instruction prior to a TLBW instruction must not generate an exception. You are recommended to use an NOP to make sure this restriction is met.
4. Three instructions are needed between MTC0 (EntryHI, EntryLo, PageMask, or Index) and subsequent TLBWI or TLBWR instructions to properly reflect the MTC0 operation.

Chapter 6
CW4011 Caches

This chapter describes the CW4011 caches and cache maintenance. It contains the following sections:

6.1 Cache Memory Organization

The CW4011 has separate caches for instructions and data: the I-cache and D-cache. The CW4011 I-cache and D-cache are organized as follows:

1. The I-cache and D-cache can be organized as direct-mapped or two-way set associative caches. A least recently used (LRU) algorithm is used in two-way set associative cache replacement for the I-cache; the D-cache uses a Random algorithm for the same.
2. The cache controllers support configurations of 1, 2, 4 or 8 Kbytes for each set. Thus, the smallest supported configuration is a 1-Kbyte direct-mapped cache, and the largest is a 16-Kbytes two-way set associative cache, with 8 Kbytes per set.
3. The caches are indexed with a virtual address.
4. They are tagged with a physical address tag.

5. One cache line consists of 8 words (or four doublewords) with a single word containing four 8-bit bytes. Refill address ordering is wrap-around from the missing address.
6. The D-cache supports both WriteBack and WriteThrough modes. If the system has no memory management unit (MMU), the WB bit in the CCC register defines the mode for all cacheable regions of memory. When the WB bit is set to zero, the mode is WriteThrough. When it is set to one, the mode is WriteBack. If the system has an MMU, the translation lookaside buffer (TLB) entry determines the mode on a per-page basis.
7. Scratchpad RAM mode is available; it works similarly to the scratchpad RAM in the LR33300. This is discussed in more detail in Section 6.4, "Cache Scratchpad RAM Mode."

6.2 Cache States

This section describes cache states for the I-cache, WriteThrough D-cache, and WriteBack D-cache.

6.2.1 I-Cache and WriteThrough D-Cache

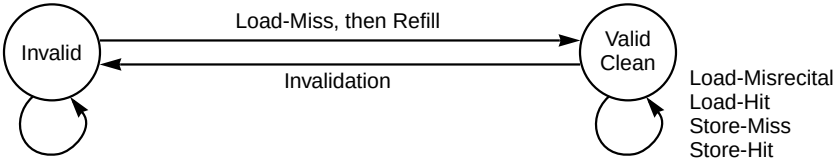
The I-cache and D-cache (when operating in WriteThrough mode) require only two states: Invalid and Valid Clean. Initialization sets all cache lines to the Invalid state. This is done using the Cache Invalidate mode described in Section 6.6.3, "Cache Maintenance by CCC Register," or the Cache Flush instructions described in Section 6.6.1, "Flush (All Cache Invalidation)."

The first time a cache line is refilled because of a cache miss, its state goes from Invalid to Valid Clean. The cache remains in the Valid Clean state until it is forced back to Invalid. This occurs in one of the following events:

- An external invalidate
- Execution of a cache flush instruction

The V bit of each cache line indicates the cache state. V = 0 is Invalid and V = 1 is Valid Clean. Figure 6.1 shows the state diagram for I-cache and WriteThrough D-cache.

Figure 6.1 Cache State Diagram—I-Cache and WriteThrough D-Cache



6.2.2 WriteBack D-Cache

When the D-cache operates in WriteBack mode, three cache line states are required: Invalid, Valid Clean, and Valid Dirty. Figure 6.2 shows the state diagram for WriteBack D-cache. The V bit and WB bit of each line indicate the state, as shown in Table 6.1.

Figure 6.2 Cache State Diagram—D-Cache WriteBack

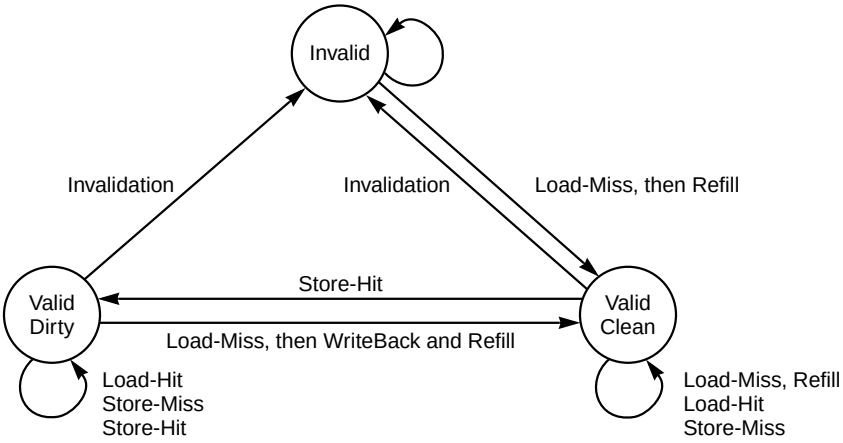


Table 6.1 D-Cache WriteBack Mode

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

A store operation is considered to be a D-cache hit when the tag is coincident with the physical address and the V bit is set. Of course, the physical address must be in a cached area.

When a Store-Miss occurs, the state condition of the cache line is not changed, and the store data is not written into D-cache. Instead, the store data is written to the four-word-deep write buffers, which pass it to the system's main memory.

Some lines, known as dirty lines, contain more recent information than the main memory. Occasionally you may need to force the writing of dirty lines to main memory. You can do this using the WriteBack Cache instruction.

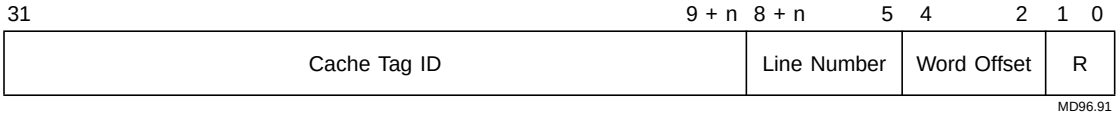
In WriteBack mode, data stored in the D-cache may not be passed on to the External Write Controller immediately. Because of this, the WriteBack cache instruction writes back each line of both sets in a two-way set associative configuration. The instruction does not check whether the address specified by the instruction would hit or miss at the cache line to which it pages. If the WB bit is set, the line data is written back and causes several stall cycles to read data from the D-cache. The actual number of stall cycles depends on the speed of memory access.

Cache lines can be invalidated by an external bus master. A cache line is invalidated when the Invalidate Address matches the Cache Tag ID, and the Cache Invalidation signal(s) are asserted.

6.3 Address and Cache Tag

Figure 6.3 illustrates the relationship between instruction and data address and cache memory location, for both direct map and two-way set associative cache configurations. The Word Offset field addresses a word in a line. The Line Number field addresses a line in the cache memory. The Cache Tag ID field serves as the tag for the address line.

Figure 6.3 Address to Cache Tag and Line Number



If the system has an MMU, the cache access is indexed by the virtual address and tagged by the physical address. Because the minimum memory page size is 4 Kbytes, there is no virtual/physical address issue if the cache set size is 4 Kbytes or less. If the cache set size is 8 Kbytes and the page size is 4 Kbytes, address bit 12 of the virtual address must be coincident with address bit 12 of the physical address.

Table 6.2 shows how the value of n determines different cache sizes.

Table 6.2 Setting Cache Size

Setting Cache Size (Kbytes)	Value of n
1	1
2	2
4	3
8	4

6.4 Cache Scratchpad RAM Mode

Both CW4011 D-cache sets and I-cache Set 1 can be configured as a scratchpad RAM. This is accomplished by setting the SR0, SR1, or ISR1 bits in the CCC register, as shown in Table 6.3.

Table 6.3 Scratchpad RAM Enables

CCC Register Bit Setting	Scratchpad Mode Enabled
SR0	D-cache Set 0
SR1	D-cache Set 1
ISR1	I-cache Set 1

A scratchpad RAM must be located in one specific physical address space like local data memory. If the CW4011 ASIC device has D-cache or I-cache tag RAMs present, the tag contents must be programmed before enabling scratchpad mode by setting the CCC register bits as follows:

- Set IsC to one
- Set TAG to one
- Clear INV to zero
- Set DE0, DE1, or IE1 to one (depending on which cache sets are to be placed in scratchpad mode)

Also, the instructions must be written into the instruction data RAM of I-cache Set 1 before the CW4011 attempts to fetch an instruction from this RAM. This is because instructions cannot be written to the instruction data RAM while scratchpad mode is enabled.

If a D-cache or I-cache RAM is only as a scratchpad RAM in an ASIC design, the cache tag RAMs can be physically removed from the device to save costs. In such a case, the D-cache tag inputs of the core must be set either HIGH or LOW, according to the address of the scratchpad RAM area. The necessary ISR1, SR0, and SR1 bits should all be always set to one.

When a cache scratchpad RAM is enabled, any accesses to the scratchpad RAM area are treated as local memory accesses without any stall cycles.

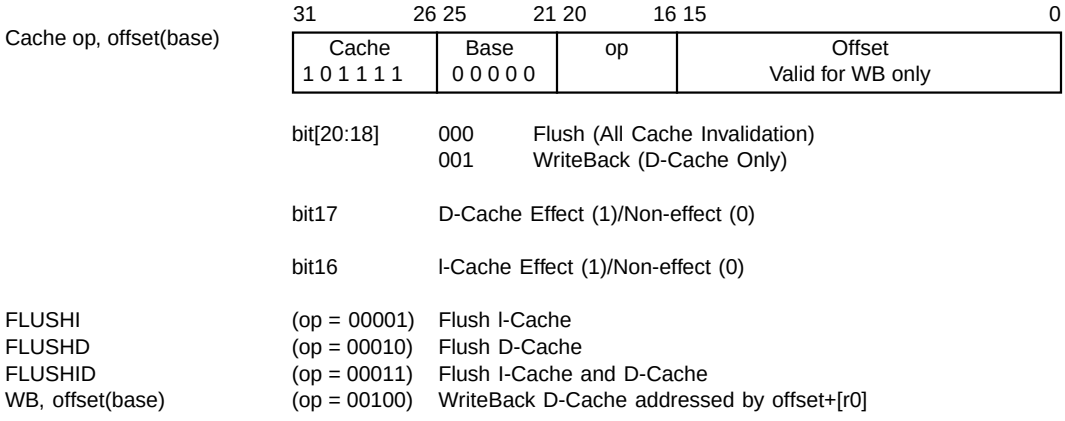
6.5 External Invalidation

I-cache and D-cache lines can be invalidated by external hardware for bus snooping. The CW4011 has an invalidate strobe and invalidate address bus input. WriteBack by external hardware is not supported. Details are described in Chapter 7, “CW4011 Signals.”

6.6 Cache Instructions

The CW4011 has two types of cache instructions for initialization and WriteBack. The cache instruction must be followed by three NOP instructions. Figure 6.4 shows the cache instruction format.

Figure 6.4 Cache Instruction Format



6.6.1 Flush (All Cache Invalidation)

One execution of a cache instruction can invalidate all lines of the D-cache, the I-cache, or both. Bit 17 of the instruction defines effect and non-effect for the D-cache, and Bit 16 defines effect and non-effect for the I-cache. If both bits are zero, this is a no operation (NOP), and the base register and the offset have no meaning.

One cache line of one or more cache sets is invalidated during one clock cycle. Invalidation starts from the WB stage of the execution pipeline, and the pipeline stall request signal is asserted during the time that the cache lines are invalidated. If the pipeline cancel signal is asserted, the invalidation is not executed. The number of the invalidation clock cycles is always 256, regardless of the cache size actually implemented. During this time, the CPU does not respond to interrupts.

6.6.2 WriteBack

WriteBack is effective for the D-cache only, so bits 17 and 16 are ignored. Bits [12:5] of the effective address, which is `offset+GPR[base]`, specify the D-cache line. Cache size is also a factor. For example, if the cache size is a 1- Kbyte direct-mapped or 2-Kbyte two-way set associative, only bits [9:5] are used and the upper bits of the effective address are ignored. Note that the tag is not checked. For more information on Cache sizing, see Appendix B, “Cache Sizing and Design Concerns.”

One WriteBack instruction writes back both lines of the two-way set associative cache if the WB bit is set. If WB is cleared, there is no operation. WB is executed at the WB stage and causes four stall cycles to read data from a dirty line. WB bits are cleared after the cache lines are written back.

6.6.3 Cache Maintenance by CCC Register

Certain CCC register bits support D-cache and I-cache maintenance and testing. Table 6.4 lists the bits of the CCC register related to the cache.

Table 6.4 CCC Bits Related to Cache Configuration

Bit(s)	Function
IE0	I-Cache Set 0 Enable
IE1	I-Cache Set 1 Enable
IS[1:0]	I-Cache Set Size (1, 2, 4, 8 Kbytes)
ISR1	I-Cache Scratchpad RAM Enable
DE0	D-Cache Set 0 Enable
DE1	D-Cache Set 1 Enable
DS[1:0]	D-Cache Set Size (1, 2, 4, 8 Kbytes)
WB	D-Cache WriteBack/WriteThrough
SR0	D-Cache Set 0 Scratchpad RAM Enable
SR1	D-Cache Set 1 Scratchpad RAM Enable
IsC	D-Cache/I-Cache Isolate Cache Mode Enable
TAG	D-Cache/I-Cache Tag Test Mode Enable
INV	D-Cache/I-Cache Invalidate Mode Enable

The CW4011 has three maintenance modes that allow you to maintain and test the internal I-cache and D-cache. The three modes are Data Test, Tag Test, and Invalidate. Before entering any of these modes, the processor must be executing in *kseg1* (noncacheable address *space0*), interrupts must be disabled, and the caches must be isolated (IsCbit = 1). When the caches are isolated, load and store instructions access the I-cache and D-cache. The system's external main memory is not affected by these load and store accesses.

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

1. Set the appropriate bits in the CCC register with IsCbit = 1. The MTC0 instruction can easily set these bits. The three instructions

immediately following the MTC0 instruction should not be load or store instructions.

The IE0, IE1, DE0, and DE1 bits in the CCC register select the cache set that is to be accessed, as shown in Table 6.5. Only one cache set should be enabled when performing a load operation. Multiple caches may be enabled when performing a store operation.

Table 6.5 TAG and INV Encoding

Bit Set	Bit Number	Cache Set Accessed
IE0	17	I-cache Set 0
IE1	16	I-cache Set 1
DE0	13	D-cache Set 0
DE1	12	D-cache Set 1

The TAG and INV bits in the CCC register select the Cache Maintenance function. Table 6.6 shows the encoding for the two bits.

Table 6.6 TAG and INV Encoding

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

2. Clear the IE bit in the Status register to disable all interrupts. This operation is usually done automatically because Cache Maintenance operations are done in an exception handler (most commonly the reset handler).

   – Data Test Mode

     In this mode, all loads and stores access the data RAMs selected by IE0, IE1, DE0, and DE1 bits. Effective lower address bits specify the cache address. The precise bit field depends on the cache size and configuration actually implemented.

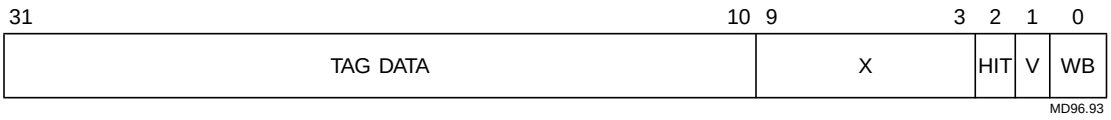
– Tag Test Mode

When TAG bit is set to one, the CW4011 is in Tag Test Mode. Load and store operations access the tag RAMs. The tag bits available for testing in the Tag Test Mode are the Tag Data, Hit, WriteBack (D-cache only), and Valid bits. Note that the WriteBack bit is present only in D-cache. The Hit bit is ignored during a store operation. For a load operation, the Hit bit is set if a match occurs.

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

A load operation from the tag RAM returns the information shown in Figure 6.5. Bits [31:10] are the tag data; bit 2 is the Hit bit; bit 1 is the Validate bit which reflects the setting of the INV bit in the CCC register; bit 0 is the WriteBack bit, which reflects the setting of the WB bit in the CCC register. You can ignore bits [9:3].

Figure 6.5 Tag Test Mode Loaded Data Format



– Invalidate Mode

When the INV bit in the CCC register is set to one, the CW4011 is in Invalidate mode. Because the caches contain random data on both warm and cold starts, software must invalidate all lines in the I-cache and D-cache. Executing store word instructions invalidates the addressed cache line in the enabled cache(s). After reset, zero must be written into all tag s for both sets of D-cache and I-cache. Cache Flush instructions can be used for the same purpose.

Chapter 7
CW4011 Signals

This chapter describes the CW4011 core I/O signals. You will find this chapter useful if you are interfacing the CW4011 with other core logic or external logic. This chapter contains the following sections:

The following signal conventions are used in this chapter:

- Active-LOW signals have a lowercase n at the end of the signal name (for example, RESETn). Active-HIGH signals have a lowercase p at the end of the signal name (for example, SCAop). Please note that some of the MDU signals do not follow this convention.
- The term assert means to drive a signal TRUE or active. The term deassert means to drive a signal FALSE or inactive.

- You can use the CW4011 core in a variety of designs and with a variety of peripheral logic. For this reason, it is not always possible to identify the agent that asserts and deasserts the I/O signals. The signal descriptions in this manual indicate the states to which the core's I/O signals must be driven. You may then select the design components needed to meet the signal requirements of the core.
- All interface signals are input to or output from the CW4011 core.

All input signals must be synchronized to the rising edge of the system clock outside the CW4011. Asynchronous signals, such as resets or interrupts, must be synchronized by at least two sequential flipflops. All output signals are synchronized to the rising edge of the system clock inside the CW4011.

7.1 CW4011 Core Signal Interfaces

The core interface signals are divided into the following eleven categories:

1. Control signals, which interface to the CP0
2. SCbus signals, which interface to the BIU
3. OCAbus signals, which interface with the LSU
4. Coprocessor signals, which interface to the ISU and LSU
5. Cache Invalidation signals, which interface to the ISU and LSU
6. Data Cache signals, which interface to the LSU
7. Instruction Cache signals, which interface to the ISU
8. WriteBack Buffer signals, which interface to the LSU
9. Memory Management Unit signals, which interface with the CP0
10. Multiply/Divide Unit (MDU) signals, which interface with the ALU
11. Miscellaneous signals, such as system clock input and endian input

Figure 7.1 illustrates the interface signal interconnections for the CW4011.

Figure 7.1 Core Interface Connections

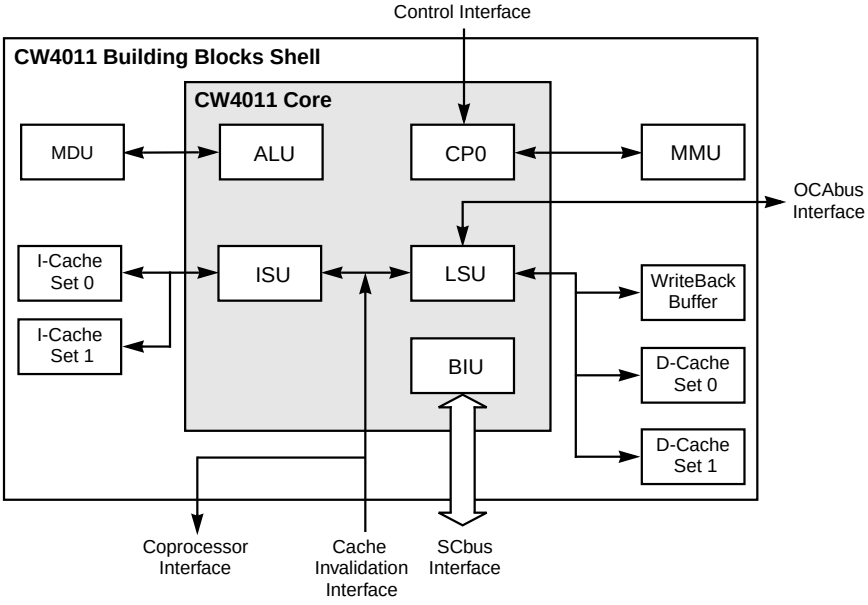


Figure 7.2 shows the CW4011 core interface signals arranged in functional groups.

Figure 7.2 CW4011 Logic Diagram

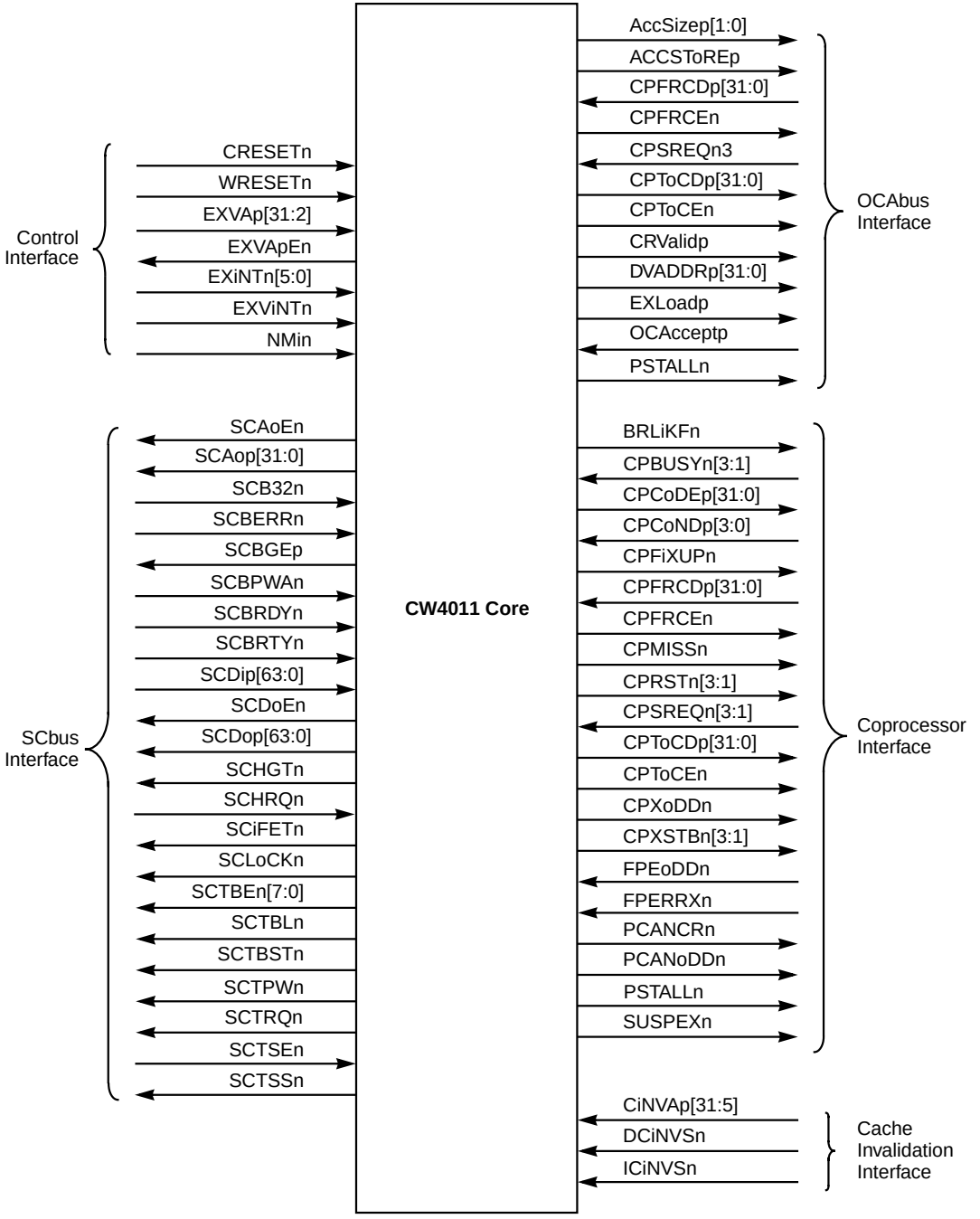


Figure 7.2 CW4011 Logic Diagram (Cont.)

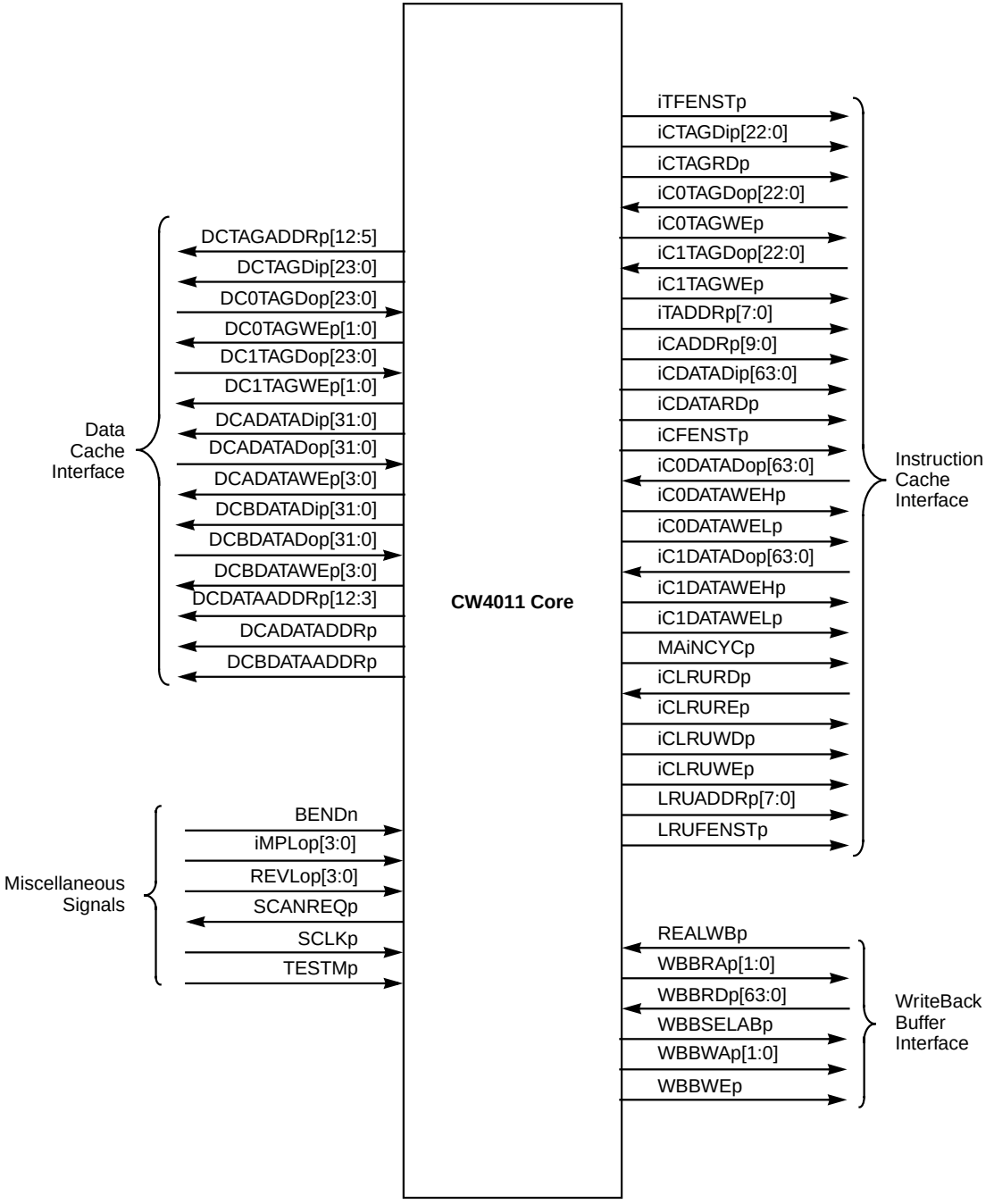
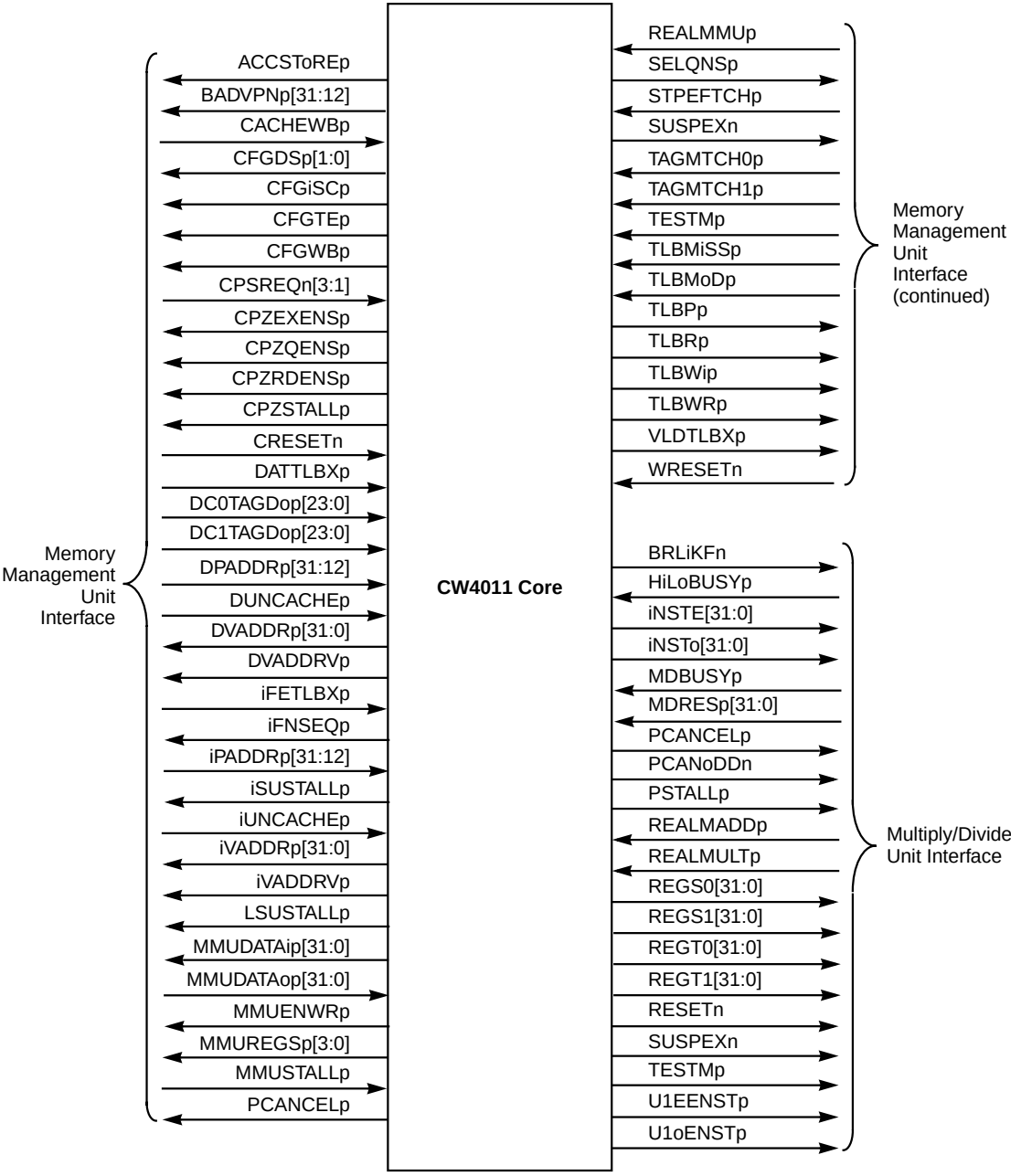


Figure 7.2 CW4011 Logic Diagram (Cont.)



7.2 Control Interface

This section describes the Reset and Interrupt signals that interface to the CP0.

CRESETn **Cold System Reset** **Input**
Asserting this signal asynchronously resets the CW4011 by initializing all internal states. CRESETn has the highest priority of all the exception inputs, and must be deasserted synchronously on the rising edge of SCLKp. When it is deasserted, the CP0 generates a cold reset exception (0xBFC00000).

EXVAp[31:2] **External Vectored Interrupt Address Input** **Input**
These signals are the interrupt vector address. They are accepted by the CW4011 when EXVApEn is asserted, and are written directly into the program counter. EXVAp[31:2] must remain stable until the EXVApEn signal is deasserted.

EXVApEn **EXVAp Enable** **Output**
The CW4011 asserts EXVApEn to enable the interrupt vector address signals (EXVAp[31:2]), and deasserts EXVApEn to disable the address signals.

EXiNTn[5:0] **External Interrupts** **Input**
External logic asserting an EXiNTn[5:0] signal causes the CP0 to generate an interrupt exception. Assertion of these inputs is indicated in the IP[7:2] field of the Cause register. Consequently, the interrupting logic should continue to assert the external interrupt input until the exception routine has serviced the interrupt.

The interrupt inputs can be individually disabled or masked by setting the appropriate bits in the Status register. External interrupts are not recognized if the interrupt enable bit in the Status register is cleared. However, the input conditions are reflected in the IP bits of the Status register. See Section 4.3.6, “Status Register,” for more information.

EXViNTn **External Vectored Interrupt Input** **Input**
EXViNTn is an external interrupt input that is driven by an external interrupt controller. See Section 4.4.6.3, “External Vectored Interrupt Exception,” for further information.

NMin **Nonmaskable Interrupt** **Input**
NMin is a nonmaskable interrupt. When the CW4011 detects that NMin is asserted, the CP0 generates a nonmaskable interrupt exception (0xBFC00000).

WRESETn **Warm System Reset** **Input**
To perform a warm reset, WRESETn must be asserted and then deasserted synchronously on the rising edge of the SCLKp. While asserted, internal states are initialized; when deasserted, the CP0 generates a warm reset exception (0xBFC00000).

7.3 SCbus Interface

This section describes the SCbus interface signals, which interface with the BIU.

SCAoEn **Address Output Enable** **Output**
When asserted, SCAoEn indicates that the address output bus SCAop[31:0] lines are valid. The CW4011 asserts this signal when the BIU is performing an SCbus transaction, and the signal remains active throughout the operation. SCAoEn also enables SCTBSTn, SCTBEn, and SCTPWn.

SCAop[31:0] **Address Output Bus** **Output**
SCAop[31:0] is the address output bus for instruction fetch and data read/write operations. The SCAop[31:0] bus is valid only when the address output enable signal (SCAoEn) is asserted. It remains valid throughout the operation until SCBRDYn, SCBRTYn, or SCBERRn is asserted.

SCB32n **32-bit Bus Width Sizing** **Input**
When asserted, SCB32n indicates that the external bus slave on the SCbus needs 32-bit bus sizing. The CW4011 samples this signal on the rising edge of the clock that synchronizes SCBRDYn. If the signal is

asserted for a 64-bit transaction, which is a doubleword or a part of a burst transaction, the BIU generates a subsequent 32-bit word transaction. The BIU also packs data to 64 bits for a read transaction or unpacks data to 32 bits for a write transaction.

SCBERRn **Bus Error** **Input**

SCBERRn is asserted to terminate the current transaction when a bus error occurs. If SCBRDYn or SCBRTYn is asserted at the same time as SCBERRn, SCBERRn has higher priority. Asserting SCBERRn causes the CP0 to generate an exception.

SCBGEp **Bus Grant Enable** **Output**

SCBGEp reflects the value of the BEG bit in the CCC register. When SCBGEp is LOW, the CW4011 is not accepting bus hold requests; when HIGH, the core is accepting requests.

SCBPWAn **Bus In-Page Write Accept** **Input**

SCBPWAn indicates that the external bus slave on the SCbus will accept in-page write transactions. External logic asserts SCBPWAn and the core samples it on the rising edge of the clock that synchronizes SCBRDYn. If the SCTPWn signal is not asserted, asserting or deasserting SCBPWAn has no effect.

SCBRDYn **Bus Ready** **Input**

The system asserts SCBRDYn for one cycle when the current transaction is successfully terminated. Asserting SCBRDYn indicates that the SCbus is available for another transaction.

SCBRTYn **Bus Retry** **Input**

An SCbus slave module (usually an I/O DRAM controller) asserts this signal for one cycle to abort the current transaction before it is complete. Asserting SCBRTYn also indicates to the core that the unsuccessful transaction must be retried later. The control state goes back to the idle state, then all bus requests are arbitrated again. If there are no other higher priority requests and SCTSEn is asserted, there is one idle state between the first transaction and a retry transaction. If SCBRDYn and SCBRTYn are asserted at the same time, SCBRTYn has the higher priority.

SCDip[63:0] **Data Input Bus** **Input**

SCDip[63:0] are data bus input signals for instruction fetch and data read transactions. The CW4011 samples SCDip[63:0] on the rising edge of the clock when SCBRDYn is asserted. Byte ordering is little endian. If you are designing a big-endian system, the higher order bits, SCDip[31:0], must be swapped with the lower order bits, SCDip[63:32], outside of the core.

SCDoEn **Data Output Enable** **Output**

SCDoEn indicates that the data output signals SCDop[63:0] are valid. the CW4011 asserts SCDoEn throughout the write transaction to indicate that the current transaction is a write transaction and to enable data output.

SCDop[63:0] **Data Output Bus** **Output**

SCDop[63:0] are the data output bus signals for data write operations and for data WriteBack from the D-cache. The signals are valid throughout the write transaction. Byte ordering is little endian. If you are designing a big-endian system, the higher order bits, SCDop[31:0], must be swapped with the lower order bits, SCDop[63:32], outside of the core.

SCHGTn **Bus Hold Grant** **Output**

The BIU enters the hold state and asserts SCHGTn to indicate that it is releasing SCbus ownership because of a bus hold request (SCHRQn).

SCHRQn **Bus Hold Request** **Input**

A LOW value on SCHRQn indicates that an external bus master is requesting ownership of the SCbus. The bus hold request has the highest priority during bus arbitration. A bus hold request cannot break continuous transactions of in-page writes and burst read/write transactions if those transactions are supported by an asserted SCTSEn signal, but must wait until SCTSEn is deasserted.

SCiFETn **Instruction Fetch** **Output**

SCiFETn indicates that the BIU is fetching instruction data. While the BIU is fetching, the core drives SCiFETn LOW and outputs it to external logic.

SCLoCKn **Bus Lock** **Output**

The core asserts SCLoCKn to indicate that it wishes to lock the SCbus and restrict bus ownership. The core asserts this signal when an executed LoadLink instruction starts a read transaction in an uncached area or a WriteThrough cached area. The core executes a StoreConditional instruction just before a write transaction starts, which deasserts SCLoCKn. During the read and write transactions, the core asserts SCLoCKn continuously, preventing bus ownership from changing during one of these transactions.

An incorrect condition can exist if a StoreConditional transaction hits the D-cache in a WriteBack cached area while SCLoCKn is asserted. In this case, the core deasserts SCLoCKn without completing any bus transactions.

SCTBEn[7:0] **Byte Enables** **Output**

SCTBEn[7:0] indicates which byte positions are valid for a transaction. The core asserts only one of the signals for a byte read or a byte write transaction. It asserts all the signals for a doubleword or a burst transaction. The SCTBEn[7:0] signals are valid when the CW4011 asserts SCAoEn.

SCTBEn Signal	Valid Byte Positions
0	SCDop[7:0]
1	SCDop[15:8]
2	SCDop[23:16]
3	SCDop[31:24]
4	SCDop[39:32]
5	SCDop[47:40]
6	SCDop[55:48]
7	SCDop[63:56]

SCTBLn **Burst Last Doubleword** **Output**

The core deasserts SCTBLn while the first, second, and third doubleword of a burst transaction is being read or written. Otherwise, the core asserts SCTBLn, which is valid on the rising edge of the system clock.

SCTBSTn **Burst Transaction** **Output**

The core asserts SCTBSTn when the first doubleword of a four doubleword transaction is being moved. The core deasserts this signal after the first doubleword has transferred, or for a singleword transaction.

SCTPWn **Next Transaction Is In-Page Write** **Output**

The core asserts SCTPWn to indicate that the next transaction will be in the same DRAM page, as defined in the CCC register. The LSU write buffer checks to see if the subsequent write request is in the same page.

When the core asserts this signal, a maximum of four sequential write transactions can occur, even if an instruction fetch request or data read request is pending. If all four write transactions are performed, the core asserts SCTPWn for the first three transactions and deasserts it for the fourth transaction. The core asserts SCTPWn from the beginning of one in-page write transaction to the end of that transaction.

SCTRQn **Transaction Request** **Output**

The core asserts SCTRQn when it needs to generate a transaction regardless of the bus hold condition. The core can use this signal to deassert a bus hold request when it needs the SCbus for an instruction fetch, data read, or data write transaction.

SCTSEn **Transaction Start Enable** **Input**

SCTSEn enables or disables a new SCbus transaction. Transaction requests are arbitrated only when SCTSEn is asserted. If an idle cycle is desired between two transactions, then this signal must be deasserted and then asserted while SCBRDYn is asserted. During the time SCTSEn is deasserted, the BIU repeats the idle state.

SCTSSn **Transaction Start Strobe** **Output**

The core asserts SCTSSn for one clock cycle at the beginning of a transaction to indicate that a new transaction has begun. If the transaction lasts through one cycle and the next transaction begins immediately, the core asserts SCTSSn continuously.

7.4 OCAbus Interface

The CW4011 has an on-chip access (OCA) interface that allows on-chip modules to be accessed at the CR stage of the pipeline without involving the SCbus. This improves performance and reduces latency by reducing traffic on the SCbus. The core is the only bus master for the OCAbus, and instructions cannot be fetched through the OCAbus.

If the module that is the target of the transaction can respond in one clock cycle, there is no penalty for a read or write transaction. A read access on the SCbus has at least a four-clock penalty, and a write access is staged through a four-deep write buffer.

Please note that the OCAbus interface and the coprocessor interface share the following signals:

- CPFRCDp[31:0]
- CPToCDp[31:0]
- CPFRCEn
- CPToCEn
- CPSREQn[3:1]
- PSTALLn

See Section 7.5, “Coprocessor Interface,” for more information on coprocessor signals. The remainder of this section describes the OCA interface signals in detail.

AccSizep[1:0] OCAbus Transaction Size **Output**

These signals indicate the transaction size of an OCAbus transaction. These signals are valid at the EX stage of the pipeline, when the core asserts either EXLoadp or ACCSToREp.

AccSizep[1:0]	Transaction Size
00	One byte
01	Halfword
10	Tribyte
11	One word

Note that data alignment for byte, halfword, or tribyte operations must be performed by either customer logic or software.

ACCSToREp **OCAbus EX Stage Store Operation** **Output**

The core asserts this signal when a store instruction is being executed in the EX stage of the pipeline. The core asserts ACCSToREp to indicate that DVADDRp[31:0] and AccSizep[1:0] are valid. DVADDRp[31:0] is decoded when the core asserts ACCSToREp. If the resulting address is for a device on the OCAbus, OCAcceptp is asserted.

CPFRCDp[31:0]

Data from OCA **Input**

This bus inputs data from an OCA device to a core general-purpose register. CPFRCDp[31:0] is valid at the CR stage of the pipeline when the core asserts the data enable signal (CPFRCEn). If there are several OCA devices, the data must be externally multiplexed.

CPFRCEn **Data from OCA Enable** **Output**

The core asserts this signal at the CR stage of the pipeline when data on the input bus (CPFRCDp[31:0]) is valid. If the pipeline enters a stall condition when there is a OCA data movement instruction in the CR stage, the core asserts CPToCEn continuously until the stall condition is resolved.

CPSREQn3 **OCA Stall Request** **Input**

An OCA device can assert this signal input at the CR stage of the pipeline when it needs to request a pipeline stall. The OCA shares CPSREQn3 with coprocessor 3, if coprocessor 3 is installed. The core asserts PSTALLn immediately when CPSREQn3 is asserted.

CPToCDp[31:0]

Data to OCA **Output**

This bus outputs data to an OCA device from a general-purpose register in the core. The CPToCDp[31:0] signals are valid at the CR stage of the pipeline when the core asserts the data enable signal (CPToCEn).

CPToCEn **Data to OCA Enable** **Output**

The core asserts this signal at the CR stage of the pipeline to indicate when the data output bus (CPToCDp[31:0]) is valid. If the pipeline enters a stall condition when there is an OCA data movement instruction in the CR stage, the core asserts CPToCEn continuously until the stall condition is resolved.

CRValidp **OCAbus CR Stage Valid** **Output**

The core asserts this signal when the CR stage of a load or store instruction is valid after it has asserted EXLoadp or ACCSToREp. If the load or store instruction is cancelled, the core deasserts CRValidp and the load/store operation must be cancelled.

DVADDRp[31:0]

OCAbus Virtual Address **Output**

This is the output bus for the OCAbus virtual address. The bus holds either the source address of a load instruction in the EX stage, or the destination address of a store instruction in the EX stage. This bus is valid only during the EX stage of the pipeline when the core asserts either EXLoadp or ACCSToREp.

EXLoadp **OCAbus EX State Load Operation** **Output**

The core asserts this signal when a load instruction is being executed in the EX stage of the pipeline. It asserts EXLoadp at the EX stage of the pipeline to indicate that DVADDRp[31:0] and AccSizep[1:0] are valid. DVADDRp[31:0] is decoded when the core asserts EXLoadp. If the resulting address is for a device on the OCAbus, OCAcceptp is asserted.

OCAcceptp **OCAbus Transaction Accepted** **Input**

The OCA module asserts this signal when it is ready to accept an OCA transaction. OCAcceptp is an output from the DVADDRp address decoder and it is asserted at the CR pipeline stage. When OCAcceptp is asserted for a read operation, the LSU selects CPFRCDp[31:0] as the data input. When OCAcceptp is asserted for a write operation, the data CPToCDp[31:0] sent to the OCA device is valid during the CR stage. The data is therefore not written into the D-cache and an SCbus transaction is not requested.

PSTALLn **Pipeline Stall Broadcasting Signal** **Output**
The core asserts this signal to indicate that all pipeline stages are stalled. This signal is valid during any stage of the pipeline.

7.5 Coprocessor Interface

This section describes the coprocessor interface signals that interface with the ISU and LSU. Contact LSI Logic if your design requires additional coprocessors (other than CP0).

BRLiKFn **Branch Likely if Even Slot is False** **Output**
The core asserts BRLiKFn when a Branch Likely instruction is in an even slot and the branch is not taken. If, at this time, a coprocessor has a valid instruction in the EX stage, the instruction must be cancelled. It is not necessary to check whether the instruction in the EX stage is in an even or odd slot, since the core asserts BRLiKFn only when the Branch Likely instruction is in the even slot. If the Branch Likely instruction in the even slot is not taken, the instruction in the odd slot must be nullified, even if it has started.

CPBUSYn[3:1]
Coprocessor Busy **Input**
These inputs are asserted when an external coprocessor is busy and cannot accept a coprocessor operation. The ISU does not assert the execution strobe signals, CPXSTBn[3:1], when the related CPBUSYn signal is asserted, and the core stalls until the busy signal is deasserted. Each coprocessor is independent and asserts its busy signal from the EX stage. The core examines the CPBUSYn signal at the RD stage of the pipeline, on the rising edge of the system clock.

CPBUSYn Signal	Busy Coprocessor
1	CP1
2	CP2
3	CP3

CPCoDEp[31:0]

CP Instruction Code Bus **Output**

This bus outputs the entire instruction bit field at the RD stage. It is valid when the core asserts one of the CPXSTBn[3:1] lines. Although the core can execute two instructions per cycle, only one coprocessor instruction can be issued in one cycle. CPCoDEp[31:0] are the selected outputs of the even and odd instruction slots. External logic must sample the bus on the rising edge of the system clock when the core asserts the strobe signal (CPXSTBn).

It is not necessary to decode all bits of an instruction, because the execution strobe signal is a partial decoding signal.

CPCoNDp[3:0]

Coprocessor Condition **Input**

These inputs are used for the coprocessor conditional branch instruction. The core samples the inputs in the ISU at the EX stage of a conditional branch instruction. The four CPCoNDp inputs are associated with the four possible coprocessors (CP0–CP3). Since CP0 does not need a conditional input, CPCoNDp0 is used as a general-purpose condition input.

CPCoNDp Signal	Coprocessor Condition
0	CP0
1	CP1
2	CP2
3	CP3

CPFiXUPn **Data Fixup Cycle Strobe for LWCz Cache Miss** **Output**

The core asserts CPFiXUPn when correct data is output on CPToCDp[31:0] during a fix-up cycle. It asserts the signal during stall cycles because LWCz cache misses cause the pipeline to stall until the data is read.

CPFRCDp[31:0]

Data from Coprocessor **Input**

This bus inputs data from a coprocessor register to a general-purpose core register or to memory. Data on the bus is valid when core asserts the data enable signal

(CPFRCEn). The core samples CPFRCDp[31:0] at the CR stage of the pipeline. If there are several external coprocessors, the data bus must be multiplexed outside the CW4011.

CPFRCEn **Data from Coprocessor Enable** **Output**

The core asserts this signal to enable the data input bus CPFRCDp[31:0]. Coprocessors can generate the same information from the instruction code (CPCoDEp) by tracking the pipeline stage. External logic must decode the coprocessor number from CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, the core asserts CPToCEn continuously until the stall condition is resolved.

CPMiSSn **Data Cache Miss Strobe for LWCz** **Output**

The core asserts CPMiSSn at the CR stage of an LWCz instruction when a D-cache miss occurs. Data at the CR stage is not correct and the correct data is put on CPToCDp[31:0] during a later fixup cycle. The core asserts PSTALLn from the WB stage of the LWCz instruction.

CPRSTn[3:1] **Coprocessor Reset** **Output**

These outputs indicate the condition of CU[3:1] bits in the CP0 status register. If the CU bit is 0, the core asserts the corresponding CPRSTn[3:1] output. The core asserts the CPRSTn[3:1] signals when a cold reset is asserted. At this time, the CU bits are cleared. The CU bits are not cleared when a warm reset is asserted. The CPRSTn[3:1] outputs allow the system designer to use software resets for external coprocessors.

CPRSTn Signal	CU bit
1	CU[1]
2	CU[2]
3	CU[3]

CPSREQn[3:1]

Coprocessor Stall Request **Input**

The external coprocessors assert these signals when they need to request a pipeline stall. Coprocessors can assert CPSREQn[3:1] while a previous coprocessor

instruction is being executed, after decoding a coprocessor instruction, and after the RD stage. When one of the CPSREQn[3:1] signals is asserted, the core asserts PSTALLn.

CPSREQn Signal	Coprocessor Requesting Stall
1	CP1
2	CP2
3	CP3

CPToCDp[31:0]
Data to Coprocessor **Output**
This bus outputs data to a coprocessor register from a general-purpose core register or from memory. Data on this bus is valid at the CR stage of the pipeline when the core asserts the data enable signal (CPToCEn).

CPToCEn **Data to Coprocessor Enable** **Output**
The core asserts this signal to indicate when the data output bus, CPToCDp[31:0], is valid at the CR stage of the pipeline. Coprocessors can generate the same information from the instruction code (CPCoDEp) by tracking the pipeline stage. The coprocessor number must be decoded from CPCoDEp[31:0]. If the pipeline enters a stall condition when there is a coprocessor data movement instruction in the CR stage, the CW4011 asserts CPToCEn continuously until the stall condition is resolved.

CPXoDDn **Coprocessor Instruction at Odd Slot** **Output**
When the core asserts an execution strobe, it also asserts CPXoDDn at the RD stage of the pipeline to indicate that the coprocessor instruction is in the odd slot. This information must be kept in the coprocessor pipeline until the CR stage. It is used to determine whether or not the instruction should be cancelled when the cancellation signal is asserted.

CPXSTBn[3:1]
Coprocessor Instruction Execution Strobe **Output**
These strobe signals indicate the start of a coprocessor operation that involves data movement. The core asserts only one of the signals during a clock cycle. The CPXSTBn[3:1] signals are partial decoding signals for an

instruction. The ISU also uses the signals to check for resource conflicts, including coprocessor busy signals. The CPXSTBn[3:1] signals are valid at the RD stage of the pipeline.

CPXSTBn Signal	Coprocessor
1	CP1
2	CP2
3	CP3

FPEoDDn **FPU Error Exception in Odd Slot** **Input**

FPEoDDn indicates whether the instruction that caused an FPU exception (FPERRXn assertion) is in an even slot (FPEoDDn is HIGH) or odd slot (FPEoDDn is LOW) when it started at the RD stage. The core ignores the FPEoDDn signal when FPERRXn is deasserted.

When the instruction is started at an RD stage, the CPXoDDn signal informs the coprocessor that the instruction is in an even or odd slot. To handle a pipeline cancel correctly, the coprocessor must keep the instruction in its pipeline registers. To execute an FPU exception precisely, the coprocessor that asserts FPERRXn at the EX stage must drive FPEoDDn correctly according to the even/odd status of the EX pipeline stage.

FPERRXn **Floating Point Unit Error Exception** **Input**

FPERRXn is an exception input, used specifically with an FPU coprocessor. The core samples the signal at any time in the EX stage and issues a pipeline cancel signal at the CR stage, in the same way as EXiNTn. In the Cause register, exception code 15 is shown for the exception if it is the highest priority. FPERRXn can be used as a user-defined coprocessor exception input.

FPERRXn must be treated precisely. The FPU asserts FPERRXn at the EX stage of the instruction with the FPEoDDn signal assertion/deassertion. The core asserts the pipeline cancel signal at the CR stage with the correct even/odd cancel signal.

PCANCRn **Pipeline Cancel at CR stage** **Output**

When one or more exceptions occurs, the pipeline is cancelled at the CR stage and the core asserts PCANCRn. Coprocessor pipelines must be cancelled to

prevent a second execution of the coprocessor instruction under either one of the following conditions: when the coprocessor returns from an exception handler or when the coprocessor has finished executing an LWCz instruction that caused a TLB miss. The WB stage is not cancelled when PCANCRn is asserted. PCANCRn is valid at the CR stage of the pipeline.

PCANoDDn **Pipeline Cancel is for Odd Slot** **Output**

PCANoDDn is valid only when PCANCRn is asserted. This signal informs coprocessors whether the cancellation is for an odd or even slot. When the core asserts the signal, cancellation applies to the odd slot. When it is deasserts the signal, cancellation applies to both even and odd slots.

The coprocessor must track which slot it is executing in based on the CPXoDDn signal. When the core asserts both PCANCRn and PCANoDDn and the coprocessor instruction is in the odd slot, the instruction must be cancelled. When the core asserts PCANCRn and deasserts PCANoDDn, the coprocessor instruction must be cancelled regardless of which slot it is operating in.

This signal is valid at the CR stage of the pipeline.

PSTALLn **Pipeline Stall Signal** **Output**

The core asserts this signal to indicate that the entire pipeline is stalled. Coprocessor pipelines must be stalled if they are executing instructions. The core asserts PSTALLn for all pipeline stalls and for an LWCz instruction D-cache miss.

SUSPEXn **Suspend EX stage** **Output**

The ISU asserts SUSPEXn request coprocessors to suspend the instruction in the EX stage. The instruction in the EX stage must be held until the ISU deasserts SUSPEXn. Instructions in the CR and WB stages must be completed.

7.6 Cache Invalidation Interface

This section describes the cache invalidation interface signals, which interface to the ISU, LSU, and CP0.

CiNVAp[31:5] **Cache Invalidation Address Bus** **Input**
The CiNVAp[31:5] input bus is the address input bus for D-cache and I-cache invalidation. When an external bus master writes data into the main memory, the address must be checked in the D-cache and the I-cache. If the address is cached, the line must be invalidated. The core samples this bus when either DCiNVSn or ICiNVSn is asserted.

DCiNVSn **D-Cache Invalidation Strobe** **Input**
When asserted, DCiNVSn indicates the Cache Invalidation Address Bus is valid and that there is need for a D-cache snooping sequence. If the cache tag is not coincident with higher address bits, the line is not invalidated.

ICiNVSn **I-Cache Invalidation Strobe** **Input**
When asserted, ICiNVSn indicates the Cache Invalidation Address Bus is valid and that there is need for an I-cache snooping sequence. If the cache tag is not coincident with higher address bits, the line is not invalidated.

7.7 Data Cache Interface

These signals interface the CW4011 with the D-cache memory. If a design involves a one-way set associative cache, the signals for the second cache should be tied either LOW or HIGH, whichever deasserts the signal. This is also true for a no-cache configuration, where both signal sets need to be deasserted.

7.7.1 D-Cache Tag RAM Signals

DCTAGADDRp[12:5]
D-Cache Tag Address **Output**
This bus carries the lower bits (bits [12:5]) of the virtual address for data load/store operations, and is the

address offset portion of the cache block. DCTAGADDRp[12:5] addresses the Tag RAM for read, write, or update operations, and connects to the Tag RAMs of both D-cache Set 0 and D-cache Set 1. The following table lists the valid bits for different D-cache RAM sizes.

D-Cache Size (Kbytes)	DCTAGADDRp[12:5] Valid Bits
8	[12:5]
4	[11:5]
2	[10:5]
1	[9:5]

DCTAGDip[23:0]

D-Cache Tag Data In **Output**

For both D-cache Set 0 and D-cache Set 1, the core drives DCTAGDip[23:0] with the upper 22 bits of the physical address (bits [31:10]) concatenated with the valid and dirty bits. DCTAGDip[1] contains the valid bit; DCTAGDip[0] contains the dirty bit, which may be ignored when using a WriteThrough policy.

DC0TAGDop[23:0]

D-Cache Set 0 Tag Data Out **Input**

D-cache Set 0 (in a two-way set associative cache) drives these signals with the appropriate tag contents.

DC0TAGWEp[1:0]

D-Cache Set 0 Tag Write Enable **Output**

The core asserts DC0TAGWEp1 to signal the D-cache Set 0 tag RAM to write the Cache Entry Tag and the valid bit (DC0TAGDip[23:1]) into the location addressed by DCTAGADDRp[12:5]. Both DCTAGADDRp[12:5] and DCTAGDip[23:1] are valid at this time.

The core asserts DC0TAGWEp0 to signal the tag RAM to write the value of DCTAGDip0 (dirty bit) into the location addressed by DC0TAGADDRp[12:5].

DC1TAGDop[23:0]

D-Cache Set 1 Tag Data Out **Input**

D-cache Set 1 (in a two-way set associative cache) drives these signals with the appropriate tag contents.

DC1TAGWEp[1:0]

D-Cache Set 1 Tag Write Enable **Output**

When the core asserts DC1TAGWEp1, the D-cache Set 1 tag RAM writes the Cache Entry Tag and the valid bit (DC0TAGDip[23:1]) into the location addressed by DCTAGADDRp[12:5]. Both DCTAGADDRp[12:5] and DCTAGDip[23:1] are valid at this time.

When the core asserts DC1TAGWEp0, the tag RAM writes the value of DCTAGDip0 (dirty bit) into the location addressed by DC0TAGADDRp[12:5].

7.7.2 D-Cache Data RAM Signals

DCADATADip[31:0]

D-Cache Data Bank A RAM Data In **Output**

These signals output data to the D-cache data Bank A RAM data bus inputs. DCDATAADDRp[12:3] and DCADATAADDRp address the Bank A data RAMs.

DCADATADop[31:0]

D-Cache Data Bank A RAM Data Out **Input**

These signals receive data from the D-cache data Bank A RAM. DCDATAADDRp[12:3] and DCADATAADDRp address the Bank A data RAMs. The RAM should continuously output the data for the given address.

DCADATAWEp[3:0]

D-Cache Data Bank A Write Enable **Output**

These signals control the byte write enables to the D-cache data Bank A RAM. When the related DCADATAWEp bit is asserted, the RAM must write that byte of DCADATADip[31:0] into memory. If all DCADATAWEp bits are asserted, the RAM must write the entire word into memory.

For example, if only DCADATAWEp[0] is asserted, the first byte is written as normal. The other three bytes of

DCADATADip[31:0] should be ignored and the data RAM should hold the previous three bytes of data.

DCADATAWEp	DCADATADip Byte	Bits
0	First	[7:0]
1	Second	[15:8]
2	Third	[23:16]
3	Fourth	[31:24]

DCBDATADip[31:0]

D-Cache Data Bank B RAM Data In **Output**

These signals output core data to the D-cache data Bank B RAM data bus inputs. DCDATAADDRp[12:3] and DCBDATAADDRp address the data within the Bank B data RAM.

DCBDATADop[31:0]

D-Cache Data Bank B RAM Data Out **Input**

This data bus transfers signals from the D-cache Set 1 data RAM to the CW4011 core. The RAM should continuously output the data for the given address.

DCBDATAWEp[3:0]

D-Cache Data Bank B Write Enable **Output**

These signals control the byte write enable inputs to the D-cache data Bank B RAM. When the related DCBDATAWEp bit is asserted, the RAM must write that byte of DCBDATADip[31:0] into memory. If all DCBDATAWEp bits are asserted, the RAM must write the entire word into memory.

For example, if only DCBDATAWEp[0] is asserted, the first byte is written as normal. The other three bytes of DCBDATADip[31:0] should be ignored and the data RAM should hold the previous three bytes of data.

DCBDATAWEp	DCBDATADip Byte	Bits
0	First	[7:0]
1	Second	[15:8]
2	Third	[23:16]
3	Fourth	[31:24]

DCDATAADDRp[12:3]

D-Cache Data Address **Output**

This bus holds the upper 10 bits used to address the memory location within the D-cache data RAMs. Both data RAMs are addressed using an 11-bit address. DCDATAADDRp[12:3] connects to bits [11:1] of the RAM Address Bus. DCADATAADDRp connects to the least-significant bit (bit 0) of the data RAM for D-cache Bank A and DCBDATAADDRp connects to D-cache Bank B bit 0.

DCDATAADDRp[12:3] connects to the data RAM address lines of both D-cache Bank A and B.

The DCDATAADDRp[12:3] signals are valid along with DCADATAADDRp, DCBDATAADDRp, DCADATAWEp, DCBDATAADDRp, and the write enable signals.

Note that the data RAM address bus is up to 11 bits and the RAM is 32 bits wide.

DCADATAADDRp

D-Cache Data Bank A Address LSB **Output**

This signal connects to the least-significant bit of the D-cache data Bank A RAM address bus (bit 0 of the address input). The concatenation of DCDATAADDRp[12:3] and DCADATAADDRp selects which word is brought into the core from the cache.

DCBDATAADDRp

D-Cache Data Bank B Address LSB **Output**

This signal connects to the least-significant bit of the address bus of the D-cache data Bank B RAM (bit 0 of the address input). The concatenation of DCDATAADDRp[12:3] and DCBDATAADDRp selects which word is brought into the core from the cache.

7.8 Instruction Cache Interface

The signals described in this section connect the CW4011 with the I-cache memory. The descriptions assume that the system is using a two-way set associative cache referred to as I-cache Set 0 and I-cache Set 1. The correct cache configuration must be set in the CCC register, and any buses or signals not needed must be deasserted by tying them either HIGH or LOW.

The I-cache tag comparators are built into the core, so external tag comparators are needed only for the D-cache.

7.8.1 I-Cache Tag RAM Signals

iTFENSTp **Tag Fetch Enable Strobe** **Output**
The core asserts iTFENSTp to signal that it is performing a store or a read operation to one of the I-cache Tag RAMs.

iCTAGDip[22:0]
New Tag to Instruction Tag RAM **Output**
This bus carries the tag data and a valid bit. Data on this bus should be written into the location specified by iTADDRp[7:0]; I-cache Set 0 if iC0TAGWEp is asserted, and I-cache Set 1 if iC1TAGWEp is asserted.

iCTAGRDp **Tag Read** **Output**
CTAGRDp is the read enable signal for both I-cache sets. The core asserts this signal and iTFENSTp to inform the I-cache tag RAMs that they should place the data selected by iTADDRp[7:0] on the iC0TAGDop[22:0] and iC1TAGDop[22:0] buses.

iC0TAGDop[22:0]
I-Cache Tag Data Out Set 0 **Input**
The I-cache Set 0 RAM outputs the tag data onto this input bus. When both the iTFENSTp and ICDATARDp signals are asserted, the tag RAM outputs the contents of the location pointed to by iTADDRp[7:0] onto the iC0TAGDop[22:0] bus.

iC0TAGWEp **Tag Write Set 0** **Output**
If iTFENSTp is asserted and the core asserts iC0TAGWEp, this informs the I-cache Set 0 tag RAM to write the data from iCTAGDip into the memory location specified by iTADDRp[7:0]. Both iCTAGDip and iTADDRp[7:0] are valid during this write transaction.

iC1TAGDop[22:0]
I-Cache Tag Data Out Set 1 **Input**
The I-cache Set 1 RAM outputs tag data to this input bus. When both iTFENSTp and ICDATARDp are asserted, the tag RAM should output the contents of the location

pointed to by iTADDRp[7:0] onto the iC1TAGDop[22:0] bus.

iC1TAGWEp **Tag Write Set 1** **Output**
If iTFENSTp is asserted and the core asserts iC1TAGWEp, this informs the I-cache Set 1 tag RAM to write the data from iCTAGDip into the memory location specified by iTADDRp[7:0]. Both iCTAGDip and iTADDRp[7:0] are valid during this write transaction.

iTADDRp[7:0] **Tag Address** **Output**
The core drives these signals with the eight lower bits of the virtual address. iTADDRp[7:0] is used to address the I-cache tag RAM. If iTFENSTp and iCTAGRDp are asserted, I-cache Set 0 should output data selected by this address onto iC0TAGDop[22:0], and I-cache Set 1 should output data addressed by this bus onto iC1TAGDop[22:0].

iTADDRp[7:0] is also used to address I-cache Set 0 and I-cache Set 1 tag RAMs for write operations.

7.8.2 I-Cache RAM Signals

iCADDRp[9:0] **I-Cache Address** **Output**
The core drives these signals with the addresses for read/write operations to the I-cache data RAMs.

iCDATADip[63:0]
Data to I-Cache **Output**
These signals hold instructions to be written into the I-cache data RAM during a write operation.

iCDATARDp **I-Cache Read Indicator** **Output**
The core asserts this signal to indicate that the current operation to the I-cache data RAM is a read. If iCDATARDp and iCFENSTp are asserted, or MAiNCYCp is asserted, instructions from the instruction RAMs of both I-cache Set 0 and I-cache Set 1 should be placed on the iC0DATADop[63:0] and iC1DATADop[63:0] buses.

iCFENSTp **Cache Fetch Enable Strobe** **Output**
The core asserts this signal to inform the instruction RAM that a read or write operation is occurring. The RAM or glue logic should check and perform the read/write oper-

ation. Data from the core is valid on the signals and buses while iCFENSTp is asserted.

iC0DATADop[63:0]

I-Cache Data In Set 0 **Input**

The core reads instructions from I-cache Set 0 on this bus. The instruction data RAM and glue logic must provide a valid instruction before the next cycle, or an error may result.

iC0DATAWEHp

Upper Word I-Cache Write Enable Set 0 **Output**

The core asserts iC0DATAWEHp to enable the I-cache Set 0 instruction RAM to write the value on the upper 32 bits of iCDATADip[63:0] to the location selected by iCADDRp[9:0]. If iC0DATAWEHp is not asserted, the value of the higher 32 bits (bits [61:32]) of the location selected by iCADDRp[9:0] of I-cache Set 0 must remain unchanged. The write transaction occurs only if iCFENSTp or MAiNCYCp is asserted at the same time as iC0DATAWEHp.

iC0DATAWELp

Lower Word I-Cache Write Enable Set 0 **Output**

The core asserts iC0DATAWELp to enable the I-cache Set 0 instruction RAM to write the value on the lower 32 bits of iCDATADip[63:0] to the location selected by iCADDRp[9:0]. If iC0DATAWELp is not asserted, the value of the lower 32 bits (bits [31:0]) of the location addressed by iCADDRp[9:0] of I-cache Set 0 must remain unchanged. The write occurs only if iCFENSTp or MAiNCYCp is asserted at the same time.

iC1DATADop[63:0]

I-Cache Data In Set 1 **Input**

The core reads instructions from I-cache Set 1 on this bus. The instruction RAM and glue logic must provide a valid instruction before the next cycle.

iC1DATAWEHp

Upper Word I-Cache Write Enable Set 1 **Output**

The core asserts iC1DATAWELp to enable the I-cache Set 1 instruction RAM to write the value on the lower 32 bits of iCDATADip[63:0] to the location selected by iCADDRp[9:0]. If iC1DATAWEHp is not asserted, the

value of the higher 32 bits (bits [61:32]) of the location addressed by iCADDRp[9:0] must remain unchanged. The write occurs only if iCFENSTp or MAiNCYCp is asserted at the same time as iC1DATAWEHp.

iC1DATAWELp

Lower Word I-Cache Write Enable Set 1 **Output**

The core asserts iC1DATAWELp to enable the I-cache Set 1 instruction RAM to write the value on the higher 32 bits of iCDATADip[63:0] to the location selected by iCADDRp[9:0]. If iC1DATAWELp is not asserted, the value of the lower 32 bits (bits [31:0]) of the location addressed by iCADDRp[9:0] must remain unchanged. The write occurs only if iCFENSTp or MAiNCYCp is asserted at the same time as iC1DATAWELp.

MAiNCYCp **I-Cache Data Maintenance Mode** **Output**

The core asserts this signal to inform the Instruction RAM that the processor is operating in Isolate Cache Maintenance mode.

7.8.3 I-Cache Least Recently Used (LRU) RAM Signals

iCLRURDp **I-Cache LRU Read Data** **Input**

The core uses this signal to read data held in LRU memory during an LRU RAM access. The LRU RAM should drive the value addressed by LRUADDRp[7:0] onto this input, if both iCLRUREp and LRUFENSTp are asserted.

iCLRUREp **Read Strobe to LRU RAM** **Output**

The core asserts this signal to indicate that the current LRU RAM operation is a read. If both iCLRUREp and LRUFENSTp are asserted, the core reads data selected by address LRUADDRp[7:0] on iCLRURDp.

iCLRUWDp **LRU Write Data** **Output**

The core drives this signal with the data that must be written into the LRU RAM in a store operation. The core drives the store address on LRUADDRp[7:0]. The core asserts iCLRUWEp and LRUFENSTp to indicate a store operation.

iCLRUWEp **Write Strobe to LRU RAM** **Output**

The core asserts this signal to indicate that the current LRU RAM operation is a write. If both iCLRUWEp and

LRUFENSTp are asserted, data should be written from iCLRUWDp into the location selected by address LRUADDRp[7:0].

LRUADDRp[7:0]
LRU Address **Output**
The core drives these signals with the address for read/write operations to the LRU RAM.

LRUFENSTp **LRU Fetch Enable Strobe** **Output**
The core asserts LRUFENSTp to indicate that a load or store bit operation to the LRU RAM is occurring. Depending on the read (iCLRUREp) or write (iCLRUWEp) enable, the address bus, LRUADDRp[7:0], addresses the memory location for a read or write transaction.

7.9 WriteBack Buffer Interface

The CW4011 core provides a simple interface to attach a doubleword WriteBack buffer. To add a WriteBack buffer to the design, a special RAM needs to be added with glue logic that controls the I/O between the core and the WriteBack buffer. This RAM should consist of four sets of 64-bit registers and control logic. The following signals interface the core with the WriteBack buffer.

REALWBp **Real WriteBack Buffer Installed** **Input**
Asserting this signal informs the core that a fully functional WriteBack buffer is installed.

WBBRAp[1:0] **WriteBack Buffer Read Address** **Output**
These signals inform the WriteBack buffer from which buffer slot (out of the four available) it should return data. The WriteBack buffer returns data on WBBRDp[63:0].

WBBRAp[1:0]	WriteBack Buffer Slot
00	0
01	1
10	2
11	3

WBBRDp[63:0]
WriteBack Buffer Read Data **Input**
The WriteBack buffer drives these signals with the data held in the buffer slot indicated by WBBRAp[1:0].

WBBSELABp **WriteBack Buffer Order** **Output**
The core asserts this signal to inform the WriteBack buffer to flip the word order during a write transaction. The default is to write the words in XY order, with X being the most significant word, and Y being the least-significant word. Asserting WBBSELABp reverses the word order to YX.

WBBWAp[1:0] **WriteBack Buffer Write Address** **Output**
These signals inform the WriteBack buffer in which doubleword slot (out of the four available) it should store data. The WriteBack buffer should write the data into a slot only if WBBWEp is asserted. Write data arrives from D-cache Set 0 through DCADATADop and from D-cache Set 1 through DCBDATADop.

WBBWAp[1:0]	WriteBack Buffer Slot
00	0
01	1
10	2
11	3

WBBWEp **WriteBack Buffer Write Enable** **Output**
The core asserts this signal to inform the WriteBack buffer that it should write data into the buffer slot indicated by WBBWAp[1:0] at the next SCLKp edge. The word write order is determined by the WBBSELABp signal and the slot written to is determined by WBBWAp[1:0].

7.10 Memory Management Unit (MMU) Interface

The CW4011 core offers a set of signals to interface with either a user-designed MMU or an LSI Coreware Building Block. The core has a built-in Coprocessor 0 to handle virtual memory addressing and exception handling, but the user must provide a TLB and a real MMU to implement a fully functional MIPS 3000/4000 microprocessor. Please

note that exceptions may be generated anytime in the IF, Q, EX, or CR pipeline stages, but are always serviced in the CR stage.

ACCSToREp **Data Access Is a Store Request** **Output**
The core asserts this signal when a store instruction is being executed in the EX stage of the pipeline. DVADDRp[31:0], DVADDRVp, and AccSizep[1:0] are valid at the same time. If ACCSToREp is deasserted, the data access is a fetch operation.

BADVPNp[31:12]
Failing Virtual Address for TLB Exceptions **Output**
These signals output the virtual page number stored in the EntryHI and Context registers that caused a TLB exception. BADVPNp[31:12] is the upper 20 bits of the virtual address that caused the exception.

CACHEWBp **Data Access WriteBack Mode** **Input**
The MMU asserts this signal to inform the core that the associated CR stage store transaction should be completed as WriteBack instead of WriteThrough. The following table lists typical operation:

CCC Register		Memory	
TE	**WB**	**Segment**	**Mode**
X	0	*kseg0*	WriteThrough
X	1	*kseg0*	WriteBack
0	0	*kuseg*, *kseg2*	WriteThrough
0	1	*kuseg*, *kseg2*	WriteBack
1	X	*kuseg*, *kseg2*	From TLB Entry

CFGDSp[1:0] **Configuration D-Cache Set Size** **Output**
These signals output information from the CCC register indicating the D-cache set size (1, 2, 4, or 8 Kbytes). If the CCC register indicates that no cache is installed, this bus is undefined.

CFGDSp[1:0]	D-Cache Size
00	1
01	2
10	4
11	8

CFGiSCp **Configuration Isolate Cache Mode** **Output**
This signal reflects the Isolate Cache Mode Bit value from the CCC register. The core asserts CFGiSCp to inform the MMU that updates to the memory should not extend past the primary data cache and addresses are not to be translated. When the MMU receives this information, it disables the TLB.

CFGTEp **Configuration TLB Enable** **Output**
CFGTEp is the TLB Enable Bit from the CCC register. The core asserts this signal to enable the TLB, if one is installed.

CFGWBp **Configuration Cache WriteBack Mode** **Output**
This signal reflects the WriteBack Mode Bit value from the CCC register. The core asserts CFGWBp to indicate that store operations are performed with the WriteBack policy.

CPSREQn[3:1]
Coprocessor Stall Request **Input**
The external coprocessors assert these signals when they need to request a pipeline stall. Coprocessors can assert CPSREQn[3:1] while a previous coprocessor instruction is being executed, after decoding a coprocessor instruction, and after the RD stage. When one of the CPSREQn[3:1] signals is asserted, the core asserts the PSTALLn signal.

CPZEXENSp **CR Stage Strobe Enable** **Output**
The core asserts CPZEXENSp to indicate that the EX stage pipeline clock is enabled.

CPZQENSp **Q Stage Strobe Enable** **Output**
The core asserts CPZQENSp to indicate that the Q stage pipeline clock is enabled.

CPZRDENSp **RD Stage Strobe Enable** **Output**
The core asserts CPZRDENSp to indicate that the RD stage pipeline clock is enabled.

CPZSTALLp **Stall Request from CP0** **Output**
CPZSTALLp indicates that internal pipeline stages have entered a stall condition by executing a WAITI (Wait Interrupt) instruction. The core asserts CPZSTALLp when the instruction is at the WB stage of the pipeline, and the

signal remains active until the core receives an external exception (enabled external interrupt, NMin, cold reset, or warm reset).

CRESETn **Cold System Reset** **Input**
Asserting this signal asynchronously resets the MMU by initializing all internal states. CRESETn must be deasserted synchronously on the rising edge of SCLKp.

DATTLBXp **TLB Exception is for Data Access** **Input**
The MMU asserts this signal during the CR stage to inform the core that a data load/store operation has caused a TLB exception. The TLBMiSSp and TLBMoDp signals must be checked to determine the cause of the exception:

TLBMiSSp	TLBMoDp	Exception Cause
0	0	TLB Invalid
0	1	TLB Mod
1	0	TLB Miss
1	1	Illegal

DC0TAGDop[23:0]
D-Cache Set 0 Tag Data Out **Input**
The D-cache Set 0 (in a two-way set associative cache) drives these signals with the appropriate tag contents.

DC1TAGDop[23:0]
D-Cache Set 1 Tag Data Out **Input**
The D-cache Set 1 (in a two-way set associative cache) drives these signals with the appropriate tag contents.

DPADDRp[31:12]
Data Access Physical Address **Input**
The MMU drives these signals with the TLB-translated physical address. If the TLB is disabled, then the MMU/TLB passes the core the original data virtual address.

DUNCACHEp **Data Access Request Is Uncached** **Input**
The MMU asserts this signal when it detects accesses to memory segment *kseg1*, which is unmapped and uncached, based on address bits DVADDRp[31:28]. Asserting this signal informs the core D-cache controller

that it should not cache the read/write operations to the address.

DVADDRp[31:0]

Data Virtual Address **Output**

The core uses these signals to output the virtual address of a load or store instruction being executed in the EX stage. The address is valid when DVADDRVp is asserted.

DVADDRVp **Data Access Request Valid** **Output**

The core asserts this signal to inform the MMU that DVADDRp[31:0] is valid, and that the MMU should perform an address translation.

iFETLBXp **TLB Exception Is for Instruction Fetch** **Input**

The MMU asserts this signal during the CR stage if a TLB exception occurs as a result of an Instruction Fetch. The TLBMiSSp and TLBMoDp signals must be checked to determine the cause of the exception:

TLBMiSSp	TLBMoDp	Exception Cause
0	0	TLB Invalid
0	1	TLB Mod
1	0	TLB Miss
1	1	Illegal

iFNSEQp **Ifetch Access Is the Target of a Branch/Jump** **Output**

The core asserts this signal to indicate that the associated Instruction Fetch (I-Fetch) in the IF Stage of the pipeline is not word-sequential. When asserted, iFENSEQp indicates that the previous instruction, now in the Q or RD Stage, was a branch/jump instruction.

iPADDRp[31:12]

Ifetch Physical Address **Input**

The MMU drives these signals with the TLB-translated instruction fetch physical address. If the TLB is disabled, then the MMU drives back the original fetch address untranslated.

iSUSTALLp **Stall Request from ISU** **Output**

iSUSTALLp is the stall request signal from the ISU. The core asserts this signal to inform the MMU that it should halt operations.

iUNCACHEp **Instruction Fetch Data is UnCached** **Input**

The MMU asserts this signal when it detects unmapped accesses to kernel segment *kseg1* (uncached memory), or when TLB cache information so indicates for *kuseg* and *kseg2*. Asserting iUNCACHEp informs the core I-cache controller that the present IFetch is uncached.

iVADDRp[31:0]

Ifetch Virtual Address **Output**

These signals hold the virtual address of the instruction which is to be translated into a physical address.

iVADDRVp **Ifetch Request Valid** **Output**

The core asserts this signal to inform the MMU that iVADDRp[31:0] holds a valid address, and that the MMU should perform an instruction fetch address translation.

LSUSTALLp **Stall Request from LSU** **Output**

LSUSTALLp is the stall request signal from the LSU. The core asserts this signal to inform the MMU that it should halt operations.

MMUDATAip[31:0]

MMU Register Data Input Bus **Output**

This bus transfers data from the core to the MMU (or CP0). MMUDATAip[31:0] transfers data to the core registers based on MFC0 instructions.

MMUDATAop[31:0]

MMU Register Data Output Bus **Input**

This bus transfers data from the MMU (or CP0) to the core. MMUDATAop[31:0] transfers data to the MMU registers based on MFC0 instructions.

MMUENWRp **MMU Register Write Enable** **Output**

The core asserts this signal to inform the MMU that it should write the data from CPToCDp into the MMU register selected by MMUREGSp[3:0]. The MMU should latch the data into the MMU register at the beginning of the WB stage.

MMUREGSp[3:0]

MMU Data Register Select (Read/Write) **Output**

The MMU decodes MMUREGSp[3:0] to determine the target of a MMU register address read/write operation.

If the operation is a write into the MMU registers, the MMU should latch the data into the MMU registers at the beginning of the WB stage. If the operation is a read, the MMU should provide valid data during the CR stage. This signal is valid in the CR pipeline stage.

MMUREGSp[3:0]	MMU Register Selected
0000	Index
0001	Random
0010	EntryLo
0100	Context
0101	PageMask
0110	Wired
1010	EntryHi

MMUSTALLp **MMU through CP0 Stall** **Input**
The MMU asserts this signal to cause the core to stall while the MMU is servicing an iTLB translation miss.

PCANCELp **Pipeline Cancel Signal from CP0** **Output**
The core asserts this signal to inform the MMU that it should clear exception registers and pipeline information. The core asserts PCANCELp when an instruction that generates an exception enters the CR pipeline stage.

REALMMUp **Real MMU Installed Indication** **Input**
The MMU asserts this signal to inform the core that a fully functional MMU with a TLB is installed.

SELQNSp **Select Q Stage (No Swap)** **Output**
The core asserts this signal to indicate that data from the Q stage must be fed into the RD stage of the pipeline. The MMU selects the Ifetch stage data if SELQNSp is deasserted, or the Q stage data if SELQNSp is asserted. When SELQNSp is asserted, it also informs the core that signals from the Q Stage are valid.

STPEFTCHp **Stop External Fetch Signal to ISU** **Input**
The MMU asserts this signal to stop the ISU from making fetch requests to external memory. Assertion of this signal informs the ISU that the translated physical address is invalid.

SUSPEXn **Suspend EX stage** **Output**
The ISU asserts SUSPEXn to request that the MMU suspend the instruction in the EX stage. The instruction in the EX stage must be held until the ISU deasserts SUSPEXn. Instructions in the CR and WB stages must be completed.

TAGMTCH0p **D-Cache Set 0 Tag Match** **Input**
The MMU asserts this signal to inform the core that a memory access has hit in the cache because the tag information matches for D-cache Set 0. A custom MMU should include comparators that perform this function. The tag data from DC0TAGDop[23:2] should be taken from the data out of the tag RAMs and compared to the translated physical address, DPADDRp[31:12] and DVADDRp[11:10]. Note that the size of the comparators depends on the size of the tags. See Chapter 6, "CW4011 Caches," for further information.

TAGMTCH1p **D-Cache Set 1 Tag Match** **Input**
The MMU asserts this signal to inform the core that a memory access has hit in the cache because the tag information matches for D-cache Set 1. A custom MMU should include comparators that perform this function. The tag data from DC1TAGDop[23:2] should be taken from the data out of the tag RAMs and compared to the translated physical address, DPADDRp[31:12] and DVADDRp[11:10]. Note that the size of the comparators depends on the size of the tags. See Chapter 6, "CW4011 Caches," for further information.

TESTMp **Test Mode Enable** **Input**
TESTMp is used for scan chain testing. It is a static input and must be tied LOW during normal operation and tied HIGH for scan chain testing.

TLBMiSSp **TLB Miss Exception** **Input**
The MMU asserts TLBMiSSp to inform the core that the MMU exception was due to a TLB miss.

TLBMoDp **TLB Modified Exception** **Input**
The MMU asserts TLBMoDp to inform the core that the MMU exception is because of a store to a page which is not marked dirty or writable.

TLBPp **TLB Probe Request** **Output**

The CP0 asserts this signal to probe the TLB (when the probe TLB for matching entry instruction is valid in the EX stage). The TLB places probe results in the Index register.

TLBRp **TLB Read Request** **Output**

The CP0 asserts this signal to request a read transaction from the TLB (when the Read Indexed TLB Entry instruction is valid in the EX stage). The TLB places the data in the EntryHi, EntryLo, and PageMask registers.

TLBWip **TLB Write Index Request** **Output**

The CP0 asserts this signal to request an indexed write transaction to the TLB (when the Write Indexed TLB Entry instruction is valid in the EX stage). Data from the PageMask, EntryHi, and EntryLo registers is written into the TLB entry defined by the Index register.

TLBWRp **TLB Write Random request** **Output**

The CP0 asserts this signal to request a random write transaction to the TLB (when the Write Random TLB Entry instruction is valid in the EX stage). Data from the PageMask, EntryHi, and EntryLo registers is written into the TLB entry defined by the Random register.

VLDTLBXp **Valid TLB Exception in CR Stage** **Output**

The core asserts this signal to indicate that a TLB exception occurred and is being reflected to the ISU. Exceptions are handled in the CR pipeline stage. The MMU must then load the EntryHi and Context registers with the failing virtual page number.

WRESETn **Warm System Reset** **Input**

To perform a warm reset, WRESETn must be asserted and then deasserted synchronously on the rising edge of the SCLKp. While asserted, MMU internal states are initialized; when deasserted, the CP0 generates a warm reset exception.

7.11 MMU to Shell Interface

These signals interface the MMU unit to external logic, but do not directly interface with the CW4011 core.

FRCMn **Force Cache Miss** **MMU Input**
Asserting FRCMn forces a cache miss for both the I-cache and D-cache. The core treats the transaction as an access to an uncached area. FRCMn is useful for debugging the system. This is a static input, and is tied LOW for software debugging.

7.12 Multiply/Divide Unit (MDU) Interface

The following signals interface the CW4011 core with a high-performance MDU. The MDU may be either an LSI Logic building block, or a user-designed MDU.

BRLiKFn **Branch Likely If Even Slot Is False** **Output**
The core asserts BRLiKFn when a Branch Likely instruction is in an even slot and the branch is not taken. When BRLiKFn is asserted, the instruction in the odd slot must be nullified, even if it has started.

HiLoBUSYp **HI/LO Register Busy Signal** **Input**
The MDU asserts this signal to inform the core that either the HI or LO register of the MDU is busy. MTLO/HI instructions and instructions involving the HI/LO register cannot be performed in the present cycle. More specifically, the MDU asserts HiLoBUSYp to prevent these instructions from entering the EX stage.

iNSTE[31:0] **Even Instruction Code** **Output**
The core drives these signals with the instruction code for the instruction in the RD stage of the even pipeline. The MDU must decode the instruction to see if it is intended for the MDU.

iNSTo[31:0] **Odd Instruction Code** **Output**
The core drives these signals with the instruction code for the instruction in the RD stage of the odd pipeline. The

MDU must decode the instruction to see if it is intended for the MDU.

MDBUSYp **MDU Busy** **Input**
The MDU asserts this signal to inform the ISU that it is busy and cannot accept any new instructions.

MDRESp[31:0]
Multiply/Divide Instruction Result **Input**
The MDU drives these signals with the output for an instruction in the EX stage. Depending on the type of instruction, and whether the instruction involves the MDU or not, the value of MDRESp[31:0] is placed back in a general-purpose core register later in the pipeline. Instructions that return values into the core general-purpose register include (but are not limited to): MFHI, MFLO, MIN, MAX, and some of the extensions for the CW4011.

PCANCELp **Pipeline Cancel Signal from CP0** **Output**
The core asserts PCANCELp when an instruction that generates an exception enters the CR pipeline stage.

PCANoDDn **Pipeline Cancel is for Odd Slot** **Output**
This signal informs the MDU whether the cancellation is for an odd or even slot. When the core asserts the signal, cancellation applies to the odd slot. When it is deasserts the signal, cancellation applies to both even and odd slots. PCANoDDn is valid at the CR stage of the pipeline.

PSTALLp **Pipeline Stall Signal from ISU** **Output**
The core asserts this signal to indicate that the pipeline is stalled. The MDU should halt operation until this signal is deasserted. PSTALLn is also asserted for any pipeline stalls.

REALMADDp **Multiplier Supports Accumulate Operation** **Input**
REALMADDp, tied HIGH, informs the core that the MDU supports MADD and MSUBB instructions. This signal is ignored if REALMULTp is deasserted.

REALMULTp **High Performance Multiplier is Installed** **Input**
REALMULTp, tied HIGH, informs the core that a MDU is installed. If the MDU supports MADD and MSUBB Instructions, then REALMADDp must be asserted. It is tied LOW if no MDU is installed.

REGS0[31:0]

S Operand, Even Instruction **Output**

The core drives these signals with the 32-bit value of the RS register. This RS value belongs to an instruction in the EX stage of the even pipeline. The MDU must decode the instruction from iNSTE[31:0] to check if the instruction involves the MDU and if it uses the RS register. REGS0[31:0] is valid only for MDU-specific instructions.

REGS1[31:0]

S Operand, Odd Instruction **Output**

The core drives these signals with the 32-bit value of the RS register. This RS value belongs to an instruction in the EX stage of the odd pipeline. The MDU must decode the instruction from iNSTo[31:0] to check if the instruction involves the MDU and if it uses the RS register. REGS1[31:0] is valid only for MDU-specific instructions.

REGT0[31:0]

T Operand, Even Instruction **Output**

The core drives these signals with the 32-bit value of the RT register. This RT value belongs to an instruction in the EX stage of the even pipeline. The MDU must decode the instruction from iNSTE[31:0] to check if the instruction involves the MDU and if it uses the RT register. REGT0[31:0] is valid only for MDU-specific instructions.

REGT1[31:0]

T Operand, Odd Instruction **Output**

The core drives these signals with the 32-bit value of the RT register. This RT value belongs to an instruction in the EX stage of the odd pipeline. The MDU must decode the instruction from iNSTo[31:0] to check if the instruction involves the MDU and if it uses the RT register. REGT1[31:0] is valid only for MDU-specific instructions.

RESETn **Reset Signal** **Output**

The core asserts this signal to indicate that either a warm reset (WRESETn) or a cold reset (CRESETn) has occurred. The core resets by jumping to the reset exception code. Building block modules, such as the MDU, must also reset.

SUSPEXn **Suspend EX stage** **Output**
The ISU asserts SUSPEXn to request MDU to suspend the instruction in the EX stage. The instruction in the EX stage must be held until the ISU deasserts SUSPEXn. Instructions in the CR and WB stages must be completed.

TESTMp **Test Mode Enable** **Input**
TESTMp is used for scan chain testing. It is a static input and must be tied LOW during normal operation and tied HIGH for scan chain testing.

U1EENSTp **Even Instruction Targeting U1 Unit** **Output**
The core asserts this signal to inform the MDU that an even pipeline instruction is targeting the U1 unit. This instruction is in the RD stage and will move into the EX stage in the next clock cycle. Instructions targeting the U1 unit may be Multiply, Divide, or Shift operations. The building block module must decode the instruction from the even pipeline to check if it is an instruction targeting the MDU. U1EENSTp serves as a warning signal for the MDU.

U1oENSTp **Odd Instruction Targeting U1 Unit** **Output**
The core asserts this signal to inform the MDU that an odd pipeline instruction is targeting the U1 unit. This instruction is in the RD stage and will move into the EX stage in the next clock cycle. Instructions targeting the U1 unit may be Multiply, Divide, or Shift operations. The building block module must decode the instruction from the even pipe to check if it is an instruction targeting the MDU. U1oENSTp serves as a warning signal for the MDU.

7.13 Miscellaneous Signals

This section describes the miscellaneous CW4011 core signals.

BENDn **Big Endian** **Input**
BENDn is a static input and must be tied LOW for big-endian addressing and HIGH for little-endian addressing. BENDn affects the byte positions for sizing and Load/Store data alignment.

For big-endian mode, the upper 32 bits of SCDip must be swapped with the lower 32 bits, and the upper 32 bits of SCDop must be swapped with the lower 32 bits outside the core.

iMPLop[3:0] **PRId IMP_LO Bus** **Input**

These signals write to the lower four bits of the Processor Revision Identifier (PRId) register IMP number. Designers can choose to place whatever value they wish on this bus to identify their product revision. Instructions reading from the PRId register return this value for the lower four bits. The upper four bits of the IMP number are set to 0b1000 by LSI Logic.

REVLop[3:0] **PRId REV_LO Bus** **Input**

These signals write the lower four bits of the PRId register REV number. Designers can choose to place whatever value they wish on this bus to identify their production revision. Instructions reading from the PRId register will return this value for the lower four bits. The upper four bits of the REV number are set to 0b0000 by LSI Logic.

SCANREQp **Scan Debug Event** **Output**

This signal is not used in this design. You may leave this signal open (unconnected).

SCLKp **System Clock** **Input**

SCLKp is the processor system clock input. It provides basic timing for the core and determines instruction cycle times. Internal core logic operates synchronously with the rising edge of SCLKp. Since the core processor operates at 90 MHz, you must supply an 90 MHz clock. SCLKp is used for all core modules.

TESTMp **Test Mode Enable** **Input**

TESTMp is used for scan chain testing. It is a static input and must be tied LOW during normal operation and tied HIGH for scan chain testing.

Chapter 8
Interface Operation

This chapter examines various CW4011 functional timing scenarios. It does not deal with all timing cases, however, but concentrates on the major CW4011 core timing operations. For details of the operation of any signals discussed in this chapter, see Chapter 7, “CW4011 Signals.”

This chapter has the following sections:

- Section 8.1, “Reset and Exception Signals”
- Section 8.2, “SCbus Interface Behavior”
- Section 8.3, “OCAbus Interface Behavior”
- Section 8.4, “Cache Interface Behavior”

In the timing diagrams shown in this chapter, all inputs and all outputs must be synchronized to the rising edge of the system clock. All inputs require setup and hold time and all outputs have valid delay times from the clock edge to the appearance of a valid level.

8.1 Reset and Exception Signals

The CW4011 has the following reset and exception inputs that connect to Coprocessor 0:

- Cold Reset
- Warm Reset
- Nonmaskable Interrupt
- Bus Error
- Floating Point Unit Exception
- Interrupts

The above inputs must be synchronized to the rising edge of the system clock. The remainder of this section discusses each of these areas in detail, except the floating point unit exception. For more information on floating point exceptions, please contact your LSI Logic representative.

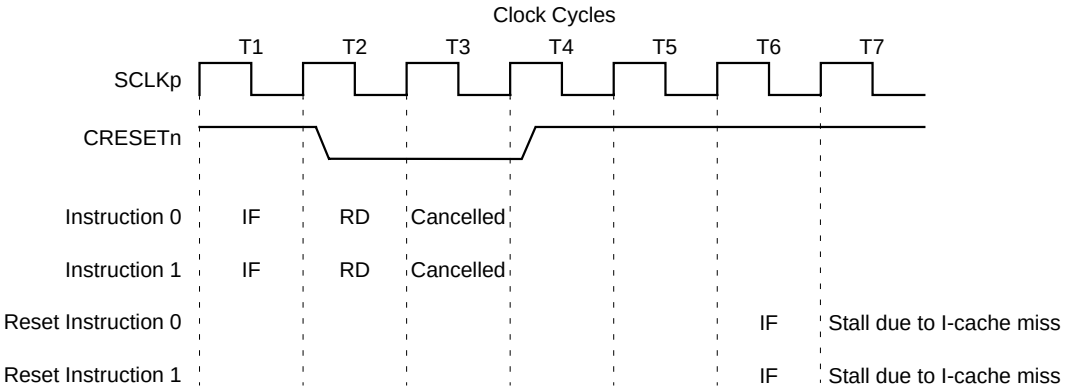
8.1.1 Cold Reset (CRESETn)

The primary purpose of a cold reset is to initialize the CW4011 core at power-up.

When asserted, CRESETn initializes the internal states and control registers in the core. CRESETn does not initialize general-purpose registers, I-cache, D-cache, or the MMU TLB.

CRESETn can be asserted asynchronously, but it must be active for at least two system clock cycles and be deasserted on the rising edge of the system clock. The CW4011 considers CRESETn a nonmaskable exception and the core is in idle mode during the period that CRESETn is asserted. Figure 8.1 shows the timing for a cold reset and the start of an instruction fetch after CRESETn is deasserted.

Figure 8.1 Cold Reset and Pipeline



8.1.1.1 Handling Cold Resets

The CPU provides a special interrupt vector (0xBFC00000) for the CRESETn exception. The reset vector resides in unmapped and uncached CPU virtual address space, so the hardware does not need to initialize the TLB or the cache to handle the exception. The processor

can fetch and execute instructions while the caches and virtual memory are in an undefined state. For further information on this subject refer to Section 4.4.5.1, “Cold Reset Exception.”

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

- In the Status register, the CU[3:0] and SR bits are cleared to zero, and the ERL and BEV bits are set to one. The other bits in the register are undefined.
- The Random register is initialized to the value of its upper bound.
- The Wired register is initialized to zero.

8.1.1.2 Servicing Cold Resets

To service the CRESETn exception, you should initialize all processor registers, coprocessor registers, caches, and the memory system. You can do this by performing diagnostic tests and by bootstrapping the operating system.

8.1.2 Warm Reset (WRESETn)

The primary purpose of the WRESETn exception is to reinitialize the processor after a fatal error.

When asserted, WRESETn initializes the CW4011 internal states and control registers. WRESETn does not initialize general purpose registers, I-cache, D-cache, or the MMU TLB.

WRESETn must be asserted and deasserted on the rising edge of the system clock. It must remain active for at least two system clock cycles. WRESETn is a nonmaskable exception and the CW4011 is in idle mode during the period WRESETn is asserted. The start of the instruction fetch after WRESETn is deasserted is the same as that of CRESETn, as shown in Figure 8.1.

8.1.2.1 Handling Warm Resets

The reset exception vector (0xBFC00000) is used for the WRESETn exception. The reset vector resides in unmapped and uncached CPU virtual address space, so the hardware does not need to initialize the TLB or the cache to handle the exception. The SR bit of the Status

register is set to distinguish the WRESETn exception from the CRESETn exception.

Unlike a nonmaskable interrupt, WRESETn resets bus state machines. Like CRESETn, it can be used on the processor in any state.

The contents of all registers are preserved when WRESETn occurs, except for the following:

- ErrorPC register, which contains the restart PC
- ERL and BEV bits of the Status register, which are set to one
- SR bit of the Status register, which is set to one

Because WRESETn can abort cache and bus operations, cache and memory contents are undefined after the WRESETn exception occurs. For further information on this subject refer to Section 4.4.5.2, "Warm Reset Exception."

8.1.2.2 Servicing Warm Resets

To service the WRESETn, you should save the current processor state to use for diagnostic purposes, and also to reinitialize all processor registers, the coprocessor, and the memory system.

8.1.3 Nonmaskable Interrupt (NMin)

The Nonmaskable Interrupt input NMin must be asserted and deasserted on the rising edge of the system clock. When NMin is sampled and found to be active on the rising edge of the clock, the CP0 provides an nonmaskable exception vector (0xBFC00000). Figure 8.2 shows the timing diagram for the fastest detected case. Figure 8.3 shows the case in which NMin is not serviced immediately because of a pipeline stall. The CW4011 detects the falling edge of NMin and latches the signal until it is ready to service it.

Figure 8.2 NMin and Pipeline (Detected Immediately)

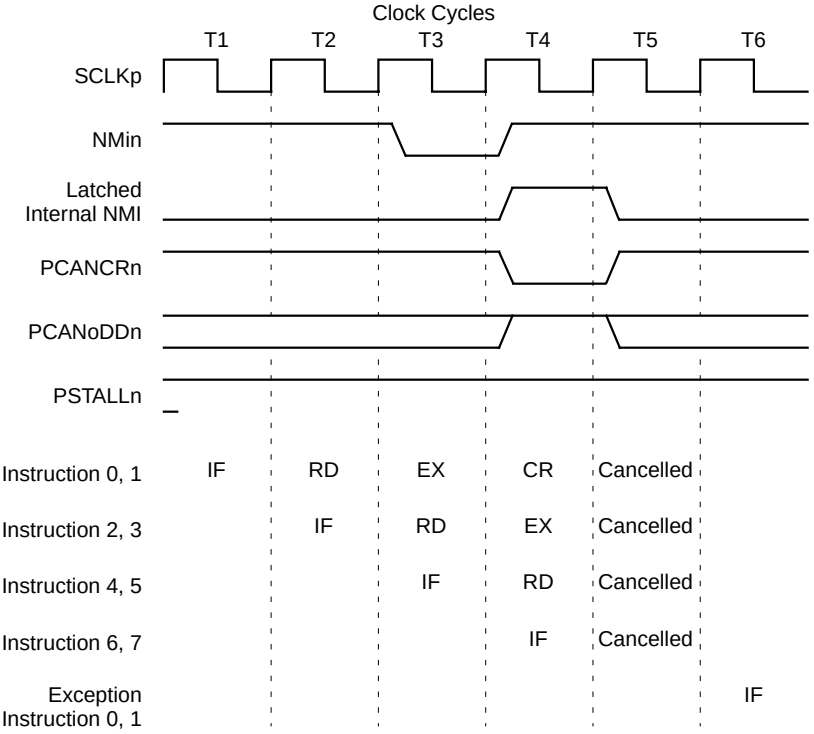
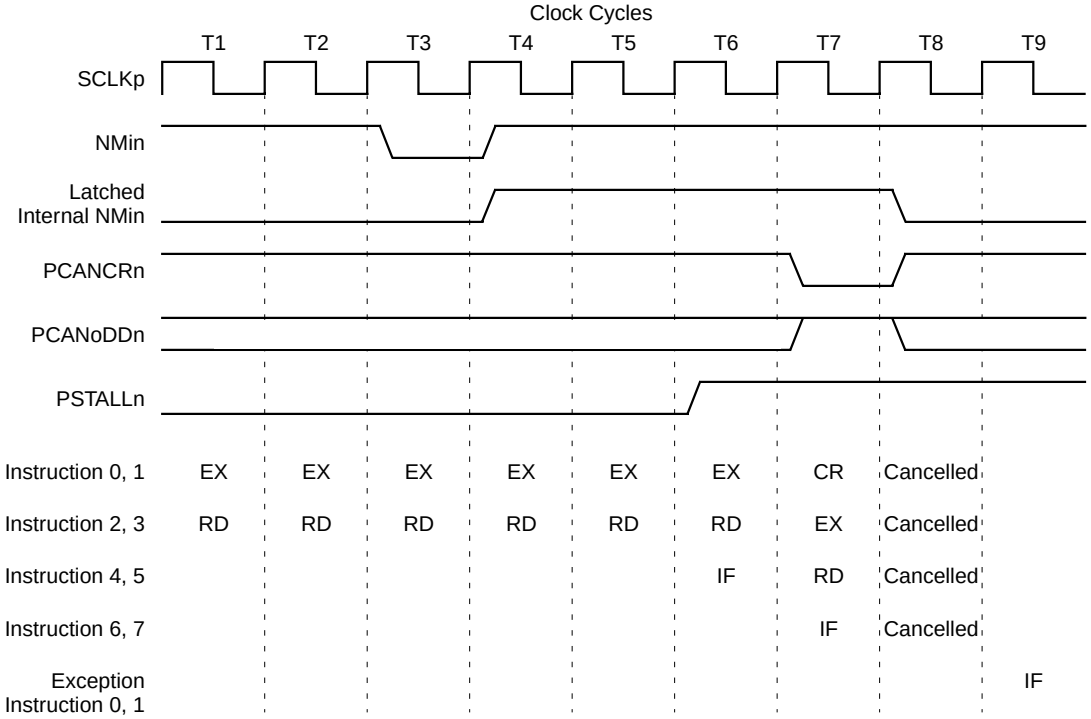


Figure 8.3 NMin and Pipeline (NMin Is Not Detected Immediately Due to Stall)



8.1.3.1 Handling a Nonmaskable Interrupt

The reset exception vector (0xBFC00000) is also used for the NMin exception. The reset vector resides in unmapped and uncached CPU address space so that the hardware does not need to initialize the TLB or the cache to handle NMin. The SR bit of the Status register is set to differentiate this exception from a CRESETn exception.

Because an NMin could occur in the middle of another exception, program execution cannot continue after NMin has been serviced.

Unlike a Cold or Warm Reset, but like other exceptions, a Nonmaskable Interrupt is taken only at instruction boundaries. The NMin exception preserves the state of the caches and memory system. For further information on this subject refer to Section 4.4.5.2, “Warm Reset Exception.”

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

- The ErrorPC register, which contains the restart PC
- The ERL and BEV bits of the Status register, which are set to one
- The SR bit of the Status register, which is set to one

8.1.3.2 Servicing a Nonmaskable Interrupt

To service the NMin exception save the current processor state for diagnostic purposes, and for reinitializing the system, including all processor registers, coprocessor registers, caches, and the memory system.

8.1.4 Bus Error (SCBERRn)

A bus error exception occurs when board-level circuitry detects events such as bus time-outs, bus parity errors, and invalid physical memory accesses. The SCBERRn exception is not maskable.

In the CW4011, bus errors are asynchronous events with respect to CPU instruction processing (much like the NMin interrupt), which means that there is no attempt to identify the instruction that was the root source of the error.

The SCBERRn input from the SCbus interface terminates a transaction and generates an exception to inform the CW4011 that an SCbus transaction has not been successfully completed. When the CW4011 is driving the SCbus, it detects the assertion of SCBERRn. SCBERRn assertion should be a synchronous one clock cycle strobe, which is latched in the CW4011 until it is serviced. Figure 8.4 shows the timing diagram in which SCBERRn is serviced immediately and Figure 8.5 shows how the exception is serviced later due to stall cycles.

Figure 8.4 Bus Error and Pipeline (Detected Immediately)

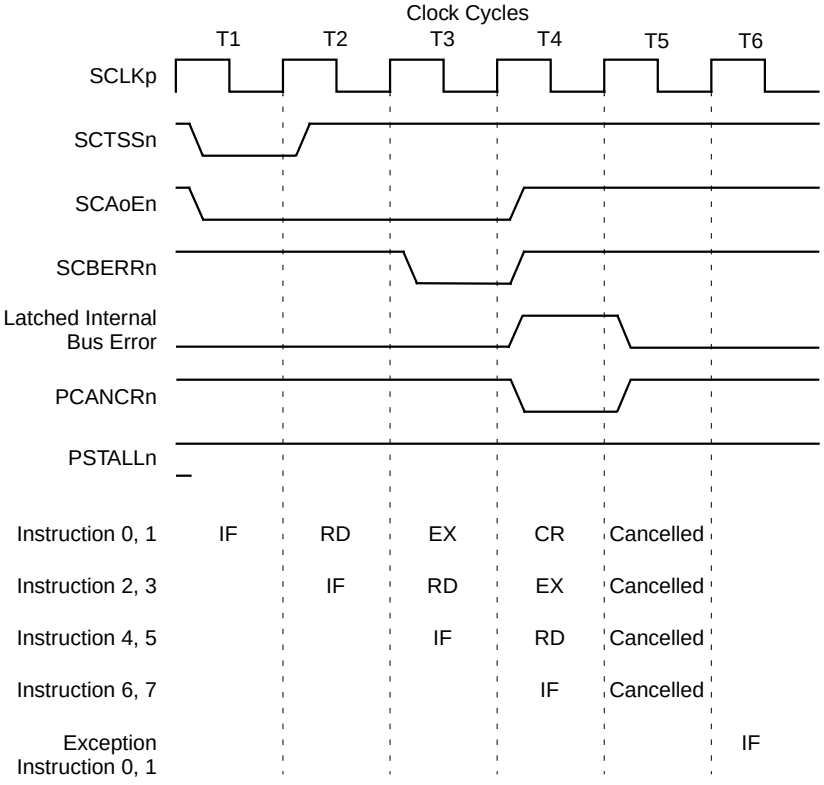
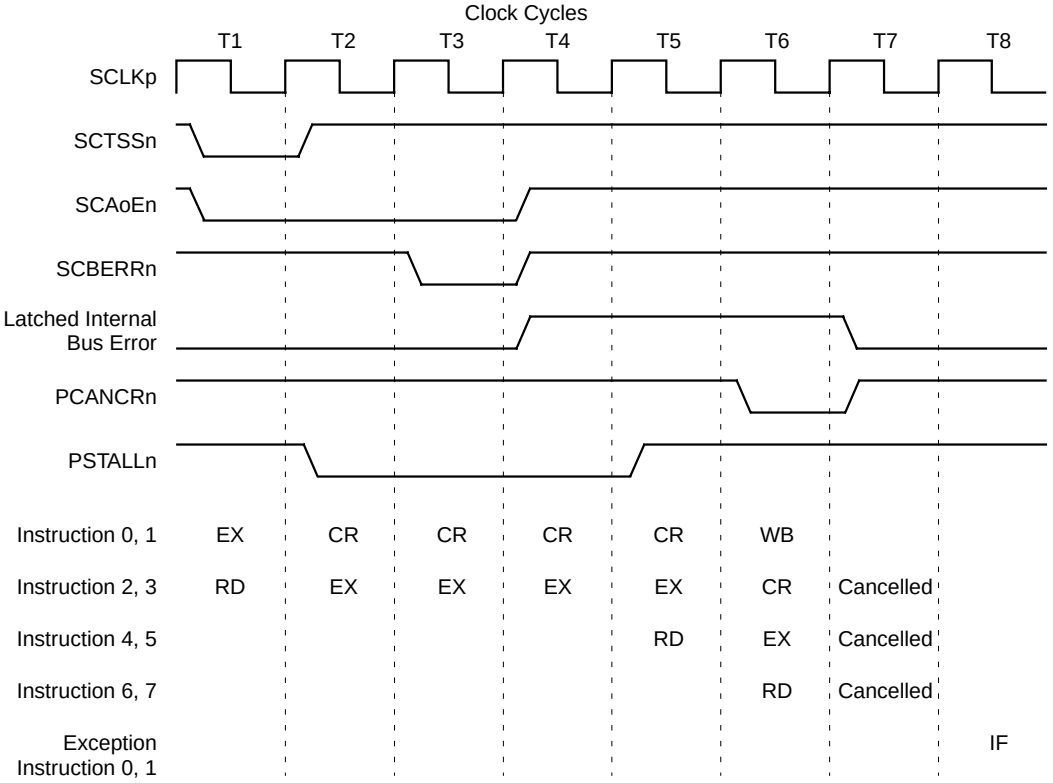


Figure 8.5 Bus Error and Pipeline (With Stall Cycles)



8.1.4.1 Handling Bus Errors

The common exception vector, shown in Table 8.1, is used for the SCBERRn exception. The ExcCode field in the Cause register is set to Bus.

Table 8.1 Common Exception Vector

Status Register	CCC Register	
DEV	**R3000 Mode**	**R4000 Mode**
0	0x80000080	0x80000180
1	0xBFC00180	0xBFC00380

The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

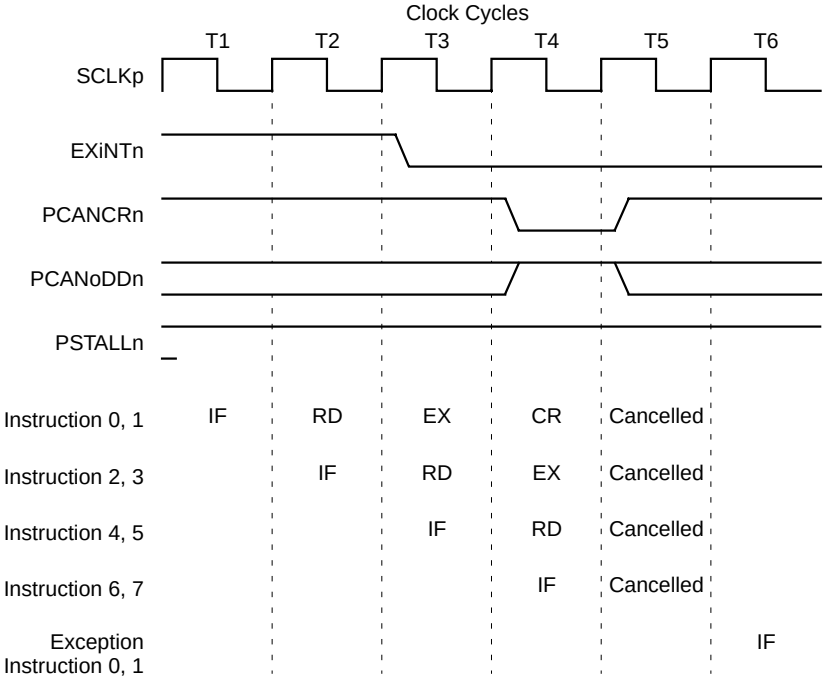
8.1.4.2 Servicing Bus Errors

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

8.1.5 External Interrupts (EXTiNTn)

The CW4011 has six external interrupt inputs, EXiNTn[5:0], which must be asserted and deasserted on the rising edge of the system clock. To mask all six external interrupts at once, you can clear the IE bit of the Status register. To mask each interrupt individually, program the INT bits in the Status register. See Section 4.3.6, “Status Register,” for further information about the Status register. The instruction fetch for the exception procedure starts two clocks after an external interrupt has been detected, provided that the pipeline is not in a stall state and there is no higher priority exception. Figure 8.6 shows the timing diagram where an interrupt is immediately detected.

Figure 8.6 Interrupt and Pipeline (Detected Immediately)



An EXTiNTn exception is similar to an NMin exception, except that external interrupts are not latched internally, and must be asserted until they are serviced. If the pipeline is in a stall cycle, the CW4011 does not service interrupts until the stall condition is resolved.

8.1.5.1 Handling External Interrupts

The common exception vector is used for the EXTiNTn exception. The ExcCode field in the Cause register is set to Int (value 0).

The IP field of the Cause register indicates the current interrupt requests. More than one of the bits may be set at the same time. None of the bits may be set if an interrupt is asserted and then deasserted before the CW4011 reads the Cause register.

The EPC register points at the first instruction for which processing was not completed unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction, and the BD bit of the Cause register is set.

See Section 4.4.6.2, “Interrupt Exception,” for further information on this subject.

8.1.5.2 Servicing External Interrupts

If one of two software generated exceptions causes the interrupt, clear the corresponding Cause register bit to zero to clear the interrupt condition.

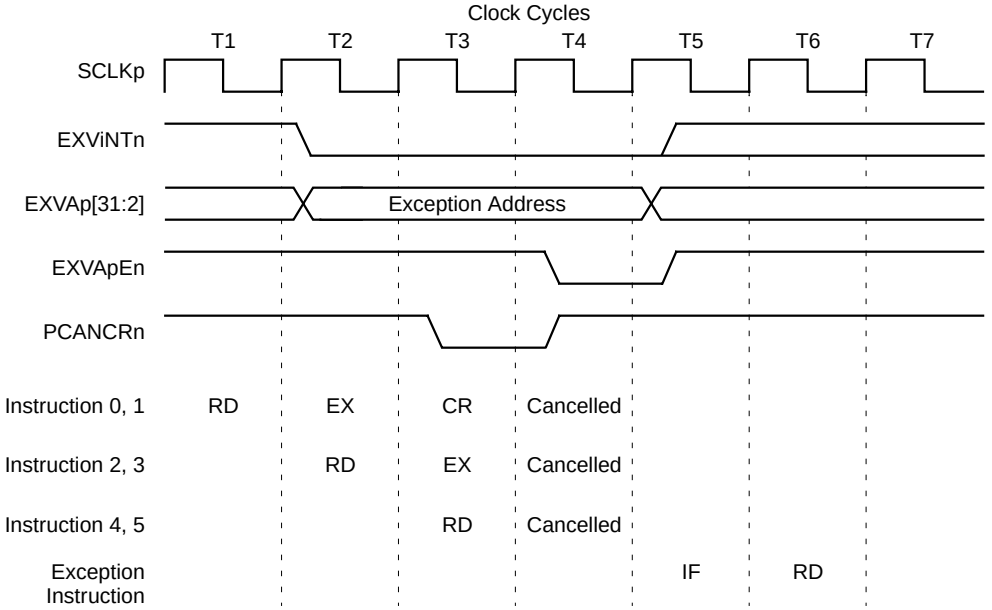
If the interrupt is hardware generated, correct the condition that caused the assertion of the interrupt pin to clear the interrupt condition.

8.1.6 External Vectored Interrupt (EXViNTn)

The CW4011 has an External Vectored Interrupt input, EXViNTn. The EXVAp[31:2] inputs provide the interrupt vector virtual address, so the common exception vector base and offset are not used.

EXViNTn must be asserted and deasserted on the rising edge of the system clock. When the EXViNTn has been sampled and found active on the rising edge of the clock, CP0 samples an exception vector from EXVAp[31:2], which is available when the enable bit EVI in the CCC is set. To mask the EXViNTn interrupt at once, you can clear the IE bit of the Status register. Figure 8.7 shows the fastest accepted case of EXViNTn. If the pipeline is stalled, it requires more clock cycles. When EXVApEn is asserted, the system may drive EXVAp[31:2].

Figure 8.7 Fastest Accepted Case of External Vectored Interrupt



8.1.6.1 Handling External Vectored Interrupts

The External Vectored Interrupt feature is available when the EVI bit in the CCC register is set.

EXViNTn has lower priority than the six external interrupts EXiNTn[5:0], but higher priority than the debug exception. To mask EXViNTn, you can use the interrupt enable bit in the Status register in a similar way to that used for external interrupts.

If EXViNTn is accepted, the CP0 reads the exception vector address on EXVAp[31:2] and writes it into the Program Counter directly. A user-defined interrupt controller provides EXVAp[31:2], so that the CW4011 jumps to the interrupt handler directly when it is requested. EXVAp[31:2] must be stable until EXVApEn is asserted. EXViNTn does not alter anything in the Cause register except the BD bit.

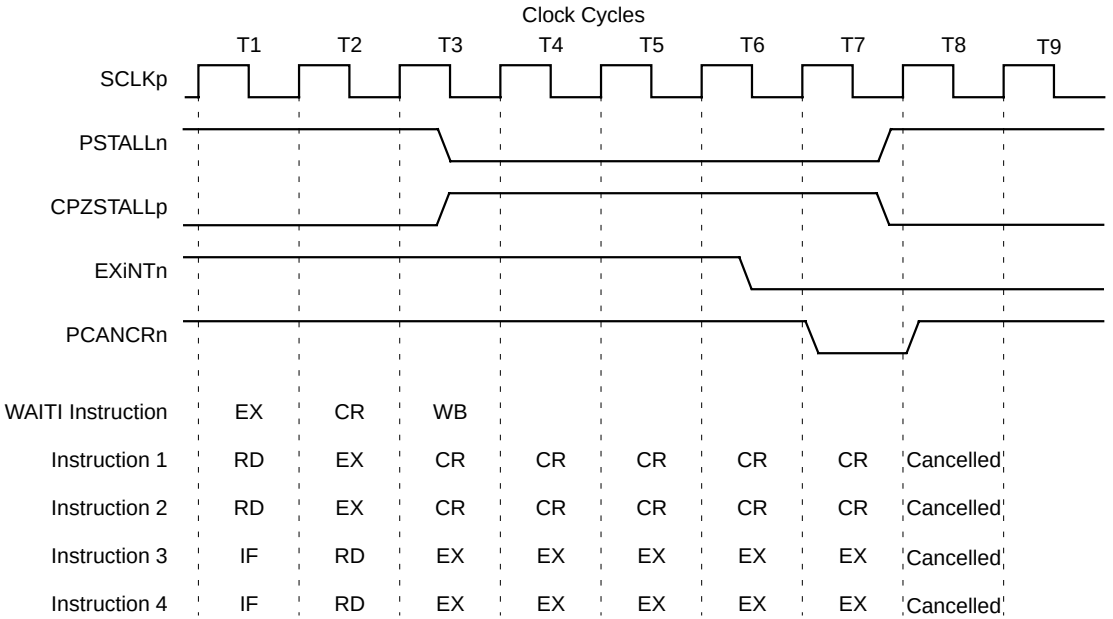
The EPC register points at the first instruction for which processing was not completed, unless this instruction is in a branch delay slot. If the instruction is in a branch delay slot, the EPC register points at the preceding branch instruction and the BD bit of the Cause register is set.

Refer to Section 4.4.6.3, “External Vectored Interrupt Exception,” for further information on this subject.

8.1.7 WAITI Instruction and CPZSTALLp

The CW4011 uses the WAITI instruction, which is one of its extended instructions, to initiate a wait state. This stalls the pipeline and reduces power consumption during the period that the CW4011 is inactive. The CW4011 wakes up when it detects an external exception input (enabled interrupt, NMin, warm reset, or cold reset). Figure 8.8 shows the timing diagram for the WAITI instruction.

Figure 8.8 WAITI and Pipeline Stall (CPZSTALLp)



The coprocessor interface signal, PSTALLn, is also asserted when the pipeline stage is in the stall condition.

At T1, the CP0 starts executing a WAITI instruction. At T3, which occurs in the WB stage of the pipeline, the CP0 requests pipeline stall and the CW4011 asserts CPZSTALLp. At T6, an external interrupt input is asserted and the CW4011 wakes up from T7. At T8, the instructions in

the pipeline stages are cancelled, and the IF stage for the exception is started from T9.

8.2 SCbus Interface Behavior

The CW4011 generates one or more external data read/write transactions on the SCbus under any of the following conditions:

- Uncached area instruction fetch
- I-cache miss
- Uncached area data read/write
- Load D-cache miss
- Any store execution in WriteThrough mode
- D-cache WriteBack

The SCbus is a flexible address/data bus. It is demultiplexed and synchronized to the system clock. It has a data width of 64 bits, but supports one type of bus sizing from a 64-bit width to a 32-bit width. The SCbus has the following transaction data sizes: byte, halfword, tribyte, 32-bit word, 64-bit doubleword, or 8-word burst (4-doubleword burst), as shown in Table 8.2.

The CW4011 has a four-line-depth write buffer for uncached, D-cache

Table 8.2 SCbus Transaction Types

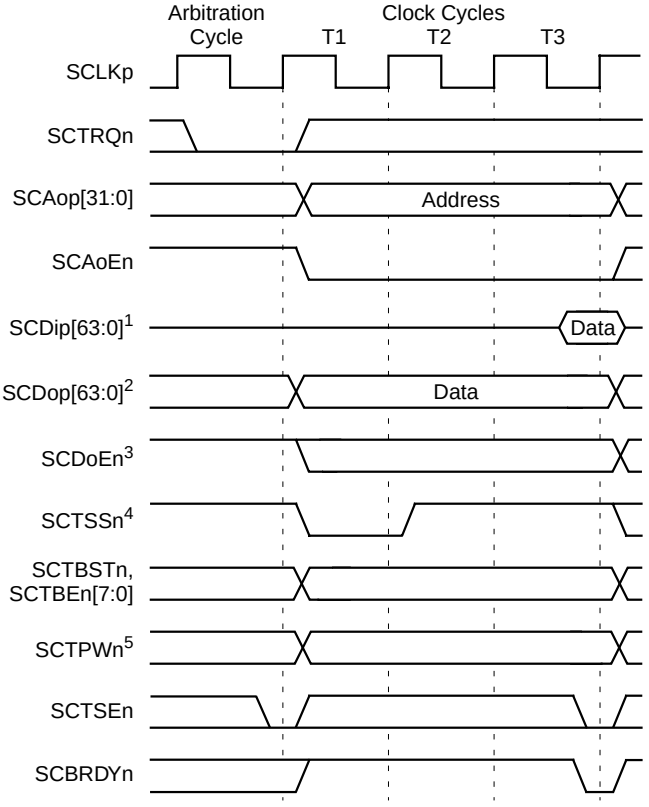
Cause of SCbus Transaction	Transaction Type	No. of Bytes
Uncached Instruction Fetch	Doubleword	8
Instruction Cache-Miss	8-word burst	32
Data Read by Uncached Load Instruction	Byte, halfword, tribyte, word	1, 2, 3, 4
Data Read by D-Cache-Miss Load Instruction	8-word burst	32
Data Write by Uncached Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by D-Cache-Miss Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by WriteThrough Store Instruction	Byte, halfword, tribyte, word, doubleword	1, 2, 3, 4, 8
Data Write by WriteBack	8-word burst	32

miss, or WriteThrough store operations. Each line in the buffer contains 32 bits of address and 64 bits of data. If word data is stored to a continuous same-doubleword alignment address, two words are stored in one line. The CW4011 then requests a doubleword write transaction on the SCbus, which the sizing function can separate into two 32-bit write transactions.

8.2.1 SCbus Basic Transaction

Figure 8.9 shows a basic SCbus transaction for a single read and write transaction. It is a three-clock-cycle transaction, which means that the SCBRDYn assertion is sampled on the rising edge of the third clock edge from the beginning of the transaction. The number of clock cycles for the fastest transaction is one clock, in which case SCTSSn is asserted continuously if the next transaction starts just after the current one. There is no limit to the maximum number of clock cycles for a transaction. A bus watchdog timer must be designed outside the core to assert the bus error signal SCBERRn, if necessary, when the transaction length is longer than the specification.

Figure 8.9 SCbus Basic Transaction



1. Read cycle.
2. Write cycle.
3. High = read, low = write.
4. Asserted at first cycle.
5. Low for in-page write.

At the beginning of a transaction, the transaction start strobe signal (SCTSSn) is asserted for one clock cycle. In addition, the address is output on the SCAop[31:0] lines and the address output enable signal (SCAoEn) is asserted to indicate that SCAop[31:0] is valid.

The Byte Enable signals (SCTBEn[7:0]) are also output. If the transaction is a four-doubleword burst, SCTBSTn is asserted during the first transaction. If the transaction is an in-page write, which means that the next transaction is in the same page, SCTPWn is asserted. It is not asserted for burst write transactions. The SCTBSTn and SCTPWn status indication signals are valid by the end of the transaction.

If the transaction is a data write, data is output to the SCDop[63:0] lines and SCDoEn is asserted from the beginning to the end of the transaction. If the transaction is a data read or instruction fetch, the SCDip[63:0] signal lines are sampled on the clock edge as the ready input SCBRDYn is asserted. SCDoEn then indicates the read/write direction of the transaction and controls the three-state buffers external to the CW4011.

Asserting SCBRDYn terminates the transaction. At the same time, the size input bus signal (SCB32n) is sampled. According to the input, the BIU of the CW4011 determines the valid byte positions for the read transaction bus sizing. If SCBRDYn is asserted for a doubleword transaction, the bus interface generates a subsequent transaction for bus sizing. The bus in-page write accept input (SCBPWAn) is also sampled in an in-page write transaction. If SCBPWAn is deasserted, the bus interface arbitrates bus requests even if the next transaction is a write transaction in the same memory page. If SCBPWAn is asserted, the bus interface does not arbitrate bus requests and the next transaction must be a write transaction in the same memory page. If SCBPWAn is asserted during the in-page write transaction but SCTSEn is deasserted, the next transaction is a write transaction in the same page.

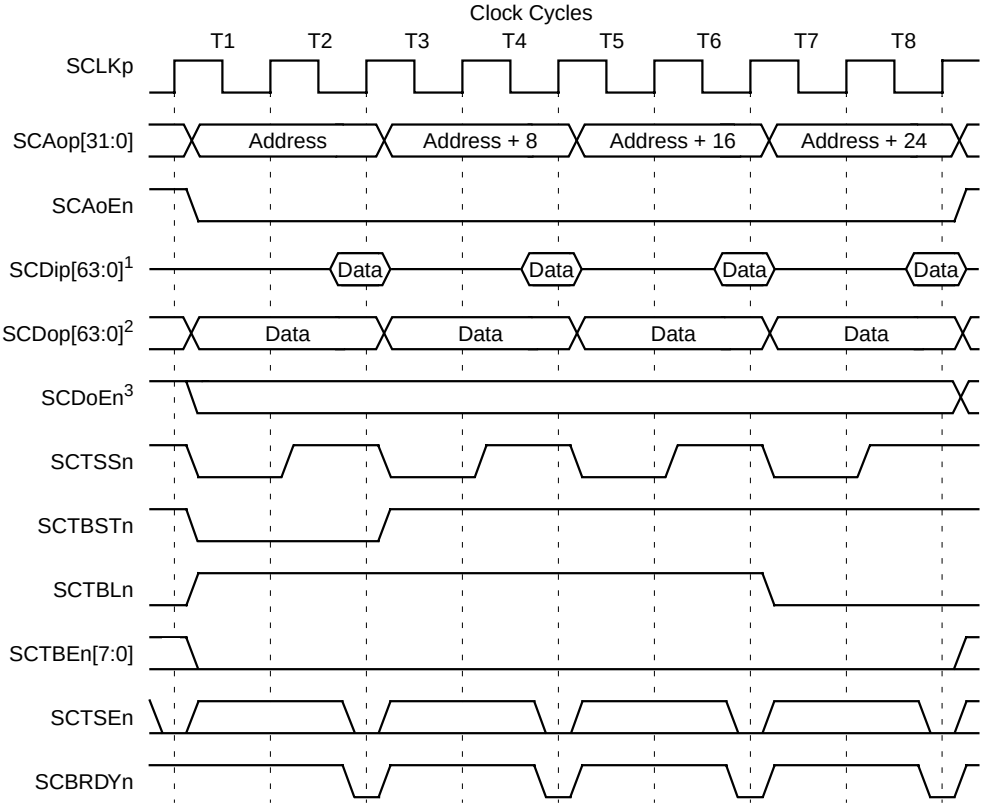
To perform an instruction fetch transaction, the CW4011 asserts SCiFETn during the same period as SCAoEn, in order to monitor the transaction.

8.2.2 SCbus Burst Transaction

When an I-cache miss occurs, the ISU requests an 8-word (4-doubleword) block burst read transaction. When a D-cache miss occurs, the LSU requests an 8-word block burst read transaction. The LSU also requests an 8-word block burst write transaction for D-cache WriteBack.

Figure 8.10 shows an eight-word burst read/write transaction that consists of four continuous transactions.

Figure 8.10 SCbus Eight-Word Burst Transaction Timing Chart



1. Read cycle.
2. Write cycle.
3. High = read, low = write.

In the first transaction, the burst transaction indicator signal, SCTBSTn, is asserted to indicate an 8-word burst transaction. Subsequent transactions are single doubleword transactions. Each transaction is terminated by an assertion of the bus ready signal, SCBRDYn. The transaction start signal, SCTSEn, is asserted for each transaction. Burst transactions can be suspended if SCTSEn is deasserted. The bus hold request signal, SCHRQn, is not accepted during a burst transaction if SCTSEn is not deasserted when SCBRDYn is asserted. SCHRQn is accepted if SCTSEn is asserted to insert one or more idle cycles when SCBRDYn is asserted.

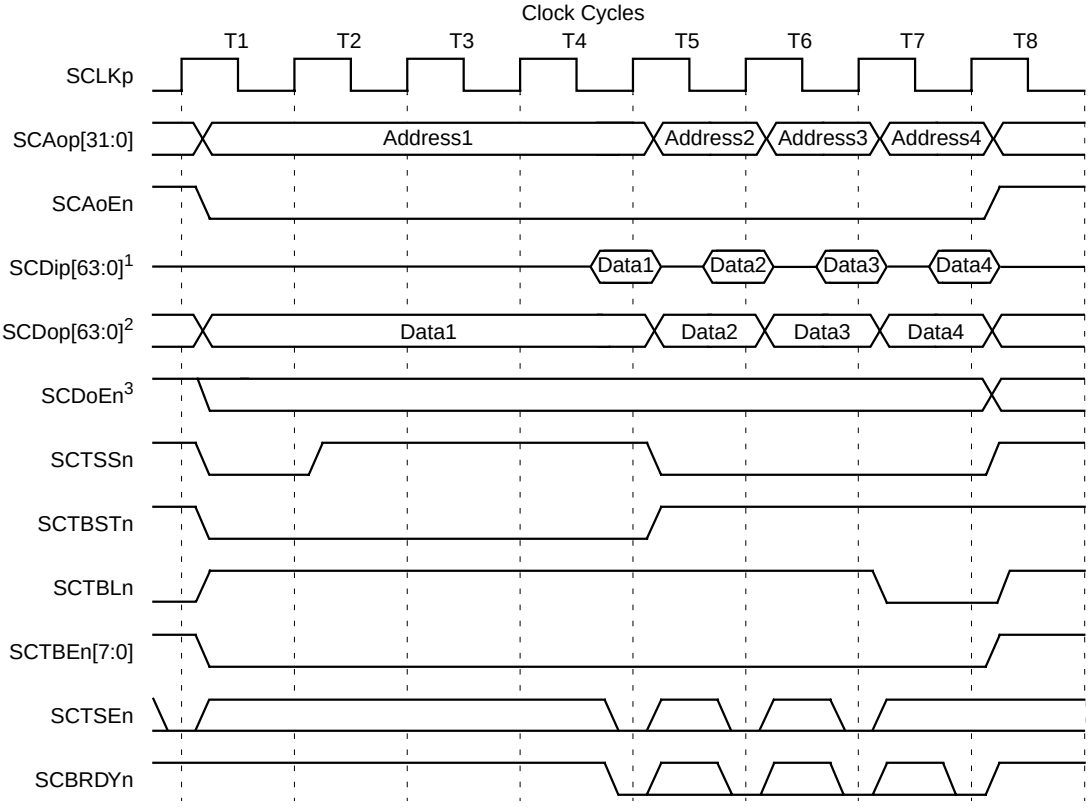
SCTBLn, which indicates whether the last transaction is a burst or a single transaction, is deasserted (HIGH) at the first, second, and third transactions of a four doubleword burst transaction.

For a burst read transaction, the first address is the missed address. The addresses of the subsequent transactions are rotative and wrap around ordering in the block. For a burst write transaction, the first address is the beginning of the block and subsequent addresses are incremental.

Bus sizing for a burst transaction is available to allow the SCbus to accomplish burst transactions to 32-bit width devices. The SCB32n input must be asserted for each group of burst transactions. If 32-bit sizing is requested for a burst transaction, eight word transactions are generated. SCTBLn is deasserted from the first to the sixth transaction. The in-page write transaction never occurs if the transaction is a burst write.

Figure 8.11 shows a timing diagram for an eight-word burst transaction. If the bus slave of the transaction is a synchronous DRAM system, there are some wait cycles for the first data transfer, but not for subsequent transfers. For a synchronous DRAM system, SCTSSn is asserted continuously for the second, third, and fourth data transfers. The DRAM controller generates addresses for these data transfers itself although SCAop also outputs addresses.

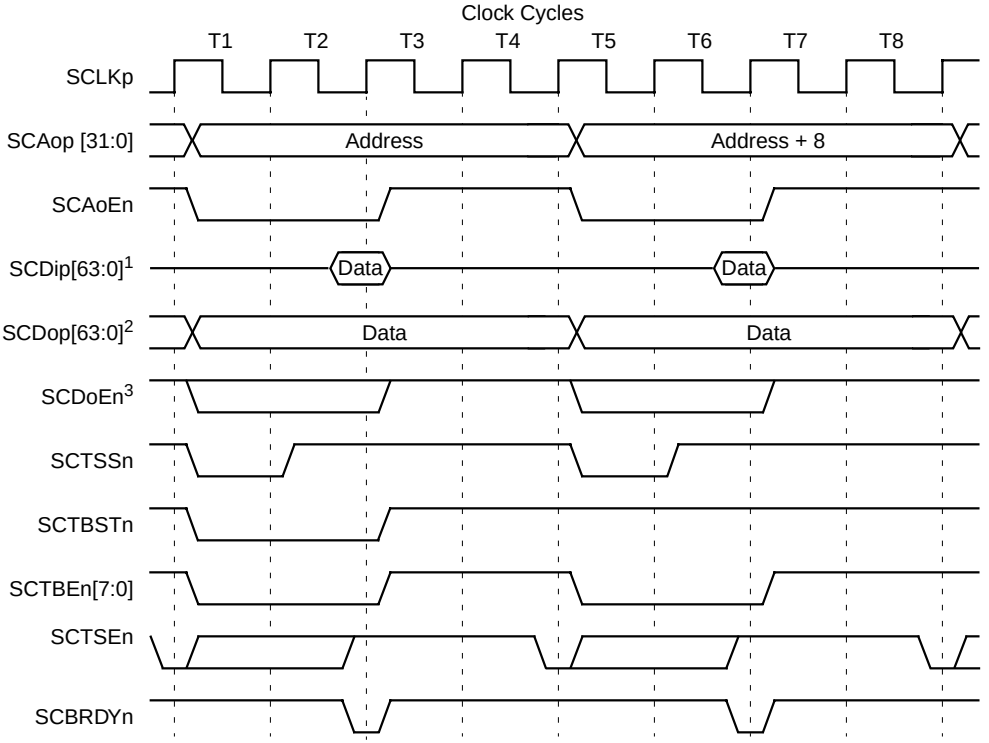
Figure 8.11 SCbus Eight-Word Burst Transaction



1. Read cycle.
2. Write cycle.
3. High = read, low = write.

Figure 8.12 shows the first and second transactions of an eight-word burst read/write. Transactions are suspended when SCTSEn is deasserted.

Figure 8.12 SCbus Eight-Word Burst Transaction



1. Read cycle.
2. Write cycle.
3. High = read, low = write.

If an individual transaction of a burst transaction is terminated with the deassertion of SCTSEn, this means the next transaction cannot proceed continuously. In that case, a hold request can be inserted. A hold request can also be inserted if a retry occurs while SCTSEn is deasserted during a burst transaction.

8.2.3 SCbus In-Page Write Transaction

An in-page write transaction is one in which continuous write accesses are made to the same row and page in a given address area. Most types of DRAM support this type of fast access, which is used to perform burst read/write transactions.

The SCbus supports continuous write transactions that have the same upper address. The external write buffer in the LSU compares upper

address bits of the current write request with those of the next write transaction in the buffer. It provides the bus interface with the result of the comparison. The address range is defined in the configuration register of the CW4011. If the two addresses have the same upper range, the in-page write output (SCTPWn) is asserted to inform the external bus slave.

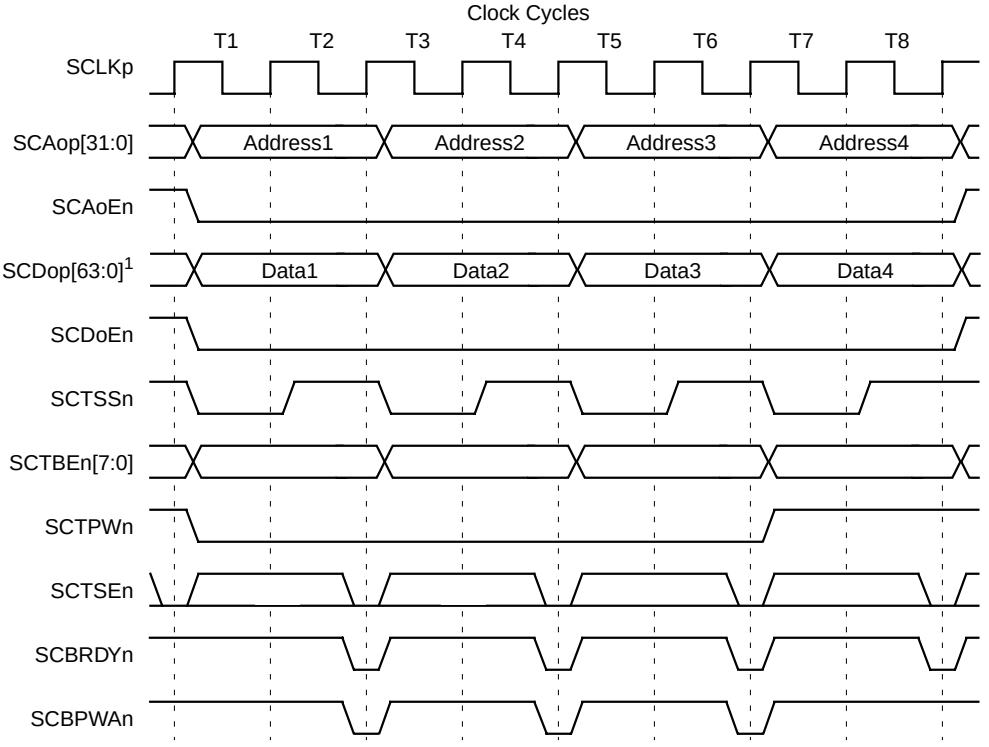
The in-page write accept input (SCBPWAn) must be asserted if the slave is able to accept in-page write transactions. If SCBPWAn is asserted, the interface does not arbitrate bus requests and the next transaction must be a write transaction. If SCBPWAn is deasserted, the bus interface performs the next transaction according to the arbitration result. SCBPWAn is sampled when the bus interface samples an assertion of the SCBRDYn signal. The bus interface performs a write transaction if SCBPWAn is deasserted and there are no higher requests. The SCBPWAn input has no meaning if the transaction is not an in-page write, and it is ignored when SCTPWn is deasserted.

The bus interface does not count the number of continuous in-page write transactions. It continues in-page writes until the write buffer is empty, a write transaction is not in the same page address area, or SCBPWAn is deasserted.

When the BIU deasserts the transaction start enable signal (SCTSEn), the CW4011 inserts one or more bus idle states between two in-page write transactions. However, the bus interface does not arbitrate requests during this idle state if the slave accepts the in-page write transactions. A hold request is allowed if the BIU deasserts SCTSEn. The bus interface does not accept the bus hold request during in-page write transactions if the BIU receives an asserted SCTSEn continuously.

Figure 8.13 shows an example of in-page write transactions.

Figure 8.13 SCbus In-Page Write Transaction (Four Words)



1. Write cycle.

8.2.4 SCbus Bus Hold

There are two ways to hold SCbus transactions:

- External logic asserts the CW4011 bus hold input, SCHRQn. The CW4011 acknowledges the request by issuing the bus hold grant signal, SCHGTn.
- External logic deasserts the transaction start enable signal, SCTSEn. Because there is no dedicated acknowledge signal associated with SCTSEn, the CW4011 deasserts the address output enable signal, SCAoEn, after the BIU deasserts SCTSEn to show that the bus interface does not own the bus.

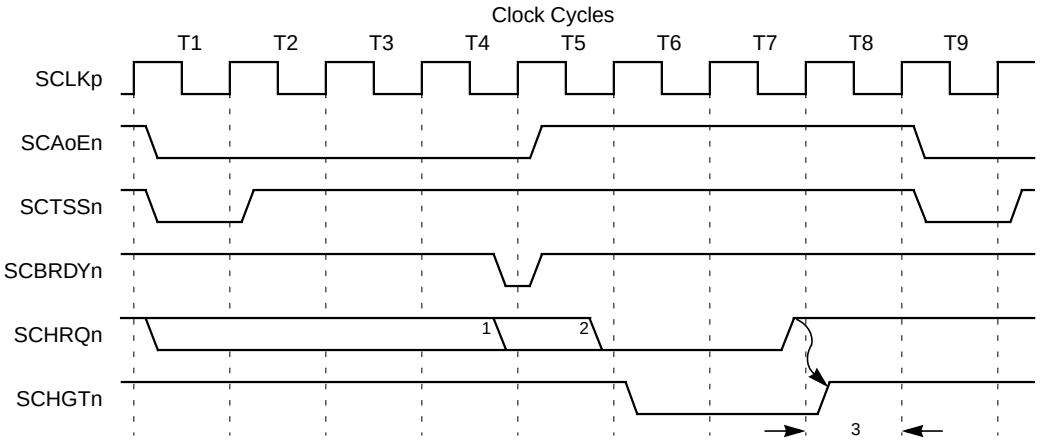
The bus hold request signal, SCHRQn, cannot break in-page write transactions and read/write burst transactions if SCTSEn is asserted

continuously. The BIU can break the transactions when it deasserts SCTSEn.

To avoid a bus deadlock, a bus retry is requested with each hold request. The current SCbus transaction generated by the BIU is then terminated by the retry and the hold request must be accepted.

Figure 8.14 shows the timing diagram for a bus hold request and the associated grant signal, SCHGTn. The CW4011 asserts the grant signal until the BIU deasserts the request. During the period the bus is held, the CW4011 does not detect bus errors.

Figure 8.14 SCbus Hold Request and Grant



1. If BIU has next transaction request, SCHRQn must be asserted before this cycle.
2. If BIU has no next transaction request, SCHGTn is asserted immediately.
3. Minimum 1 SCLK.

8.2.5 SCbus Bus Retry

The bus retry signal, SCBRTYn, is an input to the BIU. It is asserted to abort a transaction and to allow the transaction to be restarted later. The transaction state control goes to the idle state then restarts a transaction when SCTSEn is asserted. Bus retry is valid in a burst transaction. If SCBRDYn and SCBRTYn are asserted at the same time, SCBRTYn has higher priority. If SCBRTYn is asserted to hold the bus, SCHRQn should be asserted before or at the same time as SCBRTYn.

8.2.6 SCbus Bus Error

The external bus controller asserts the BIU bus error signal, SCBERRn, when the current transaction must be terminated as a bus error. If SCBRDYn is asserted at the same time, SCBERRn has higher priority.

Assertion of SCBERRn forces the CW4011 to exit the sequential transactions of in-page write and read/write burst transactions. The states of service and transaction control go to the idle state. If the transaction is a burst (cache refill or WriteBack), the CW4011 invalidates the cache line.

When a bus error occurs, the CP0 issues a bus error exception. See the Section 8.1.4, “Bus Error (SCBERRn),” for more details. A bus error exception is a fatal error for the CW4011.

8.2.7 SCbus Bus Sizing

The SCbus supports bus sizing for slaves that need sequential address access to 32-bit data. When sizing is requested, the SCB32n input to the CW4011 is asserted to separate a doubleword transaction, including part of a burst transaction, into two singleword transactions. The bus interface also selects valid byte positions for a word or a partial word transaction if SCB32n is asserted. In the case of a word or a partial write transaction, the bus interface outputs word data to both the upper and lower 32 bits of the data output bus according to address bit 2. The bus interface then completely supports a 32-bit bus interface. Although SCB32n is sampled with the assertion of the ready signal input, the bus interface behaves as a normal 32-bit data width bus if SCB32n is always asserted.

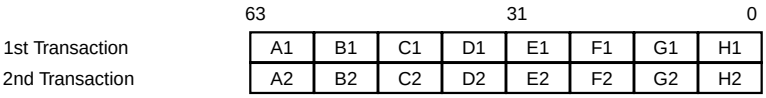
If 16-bit or 8-bit width bus sizing is needed, it must be supported outside the CW4011 core.

8.2.7.1 Read Bus Sizing

When sizing occurs at a byte, halfword, tribyte, or word during a read transaction, the CW4011 BIU can move sampled 32-bit word data to the valid position according to the setting of address bit 2. If sizing is requested for a doubleword transaction, the BIU samples 32-bit data at the first transaction then generates a subsequent transaction and packs the first 32 bits and the subsequent 32 bits. The packed data is sent to the ISU or LSU. Figure 8.15 shows the relationship between the valid

byte positions of the first and subsequent transactions. In the case of a non-doubleword read, the behavior of a byte, a halfword, and a tribyte transaction is the same as that of a word transaction because sizing supports 32-bit mode only. You can assume that the bus interface samples a doubleword (8 bytes). Figure 8.15 shows an example in which the bus interface samples a doubleword (8 bytes).

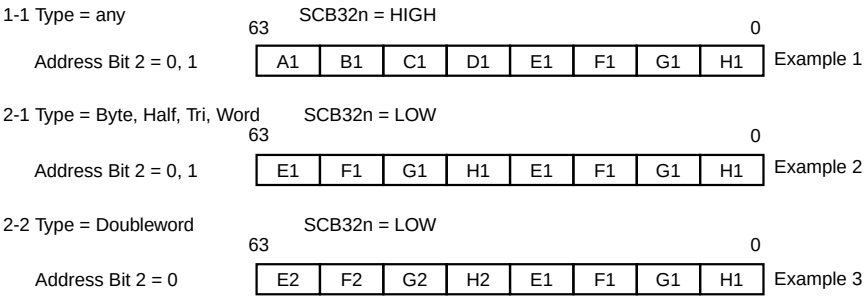
Figure 8.15 Sampled Bytes of First and Second Transaction SCbus Data



1. The 2nd transaction is generated when the transaction is a doubleword or a part of a burst with SCB32n = LOW.

If you are reading a doubleword with 32-bit bus sizing, you will need a second transaction. Figure 8.16 shows the doubleword data that is sent to the ISU or LSU.

Figure 8.16 Read Bytes to ISU and LSU with Sizing

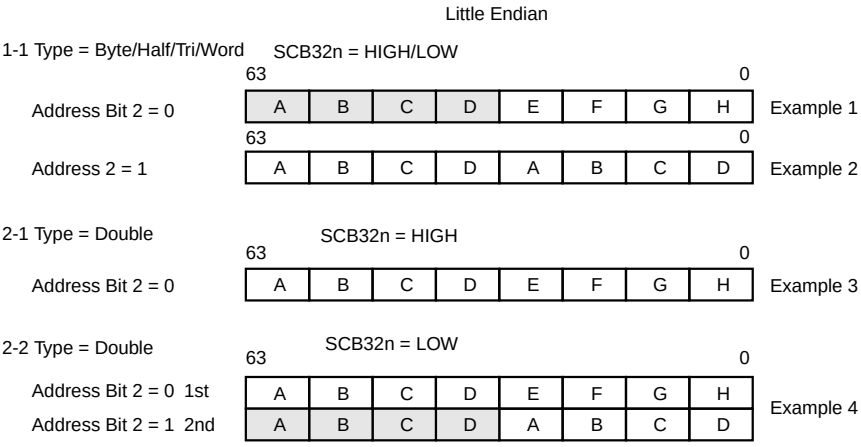


1. In Example 1, one transaction is initiated. The eight bytes sampled are transferred to the ISU and LSU without any change.
2. In Example 2, one transaction is initiated. Four bytes are sampled (bits [31:0]). They are transferred to bits [63:32] and [31:0] of the ISU and LSU.
3. In Example 3, two transactions are initiated. Bits [31:0] of the first transaction are output on bits [31:0], and bits [31:0] of the second transaction are output on bits [63:32]. This doubleword is transferred to the ISU or the LSU.

8.2.7.2 Write Bus Sizing

In the case of a non-doubleword write transaction, the bus interface selects the upper or lower 32 bits of data from the LSU and outputs the same 32-bit word data to the SCbus according to address bit 2. The data is output to the SCbus before the bus interface detects the sizing input. In the case of a doubleword write transaction, the bus interface generates a subsequent sizing transaction if the sizing input is asserted at the first transaction. Figure 8.17 shows the relationship between the doubleword data from the LSU and the SCbus. Bytes shown in the shaded area have no meaning for the SCbus write transaction.

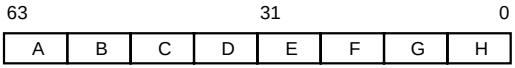
Figure 8.17 Write Bytes to the SCbus with Sizing



1. In Example 1, one transaction is initiated. A doubleword from the LSU is output on the date bus without any changes.
2. In Example 2, one transaction is initiated. Bits [63:0] from the LSU are output on bits [63:32] and [31:0] of the data bus.
3. In Example 3, one transaction is initiated. A doubleword from the LSU is output on the data bus without any change.
4. In Example 4, two transactions are initiated. In the first transaction, a doubleword from the LSU is output on the data bus without any change. In the second transaction, bits [63:32] from the first transaction are output to bits [31:0] of the data bus.

As shown in Figure 8.18, you can assume that the LSU sends a doubleword, regardless of the transaction type.

Figure 8.18 Write Data Bytes from LSU



8.2.8 SCbus Bus Lock

The CW4011 SCLoCKn output signal indicates that the SCbus is asking to lock bus ownership. The CW4011 asserts SCLoCKn when the CW4011 executes a Load Linked instruction to start a read transaction in an uncached area or WriteThrough cached area. It deasserts the signal just before it executes a Store Conditional instruction to start a write transaction. During the read write transactions, the CW4011 asserts SCLoCKn continuously.

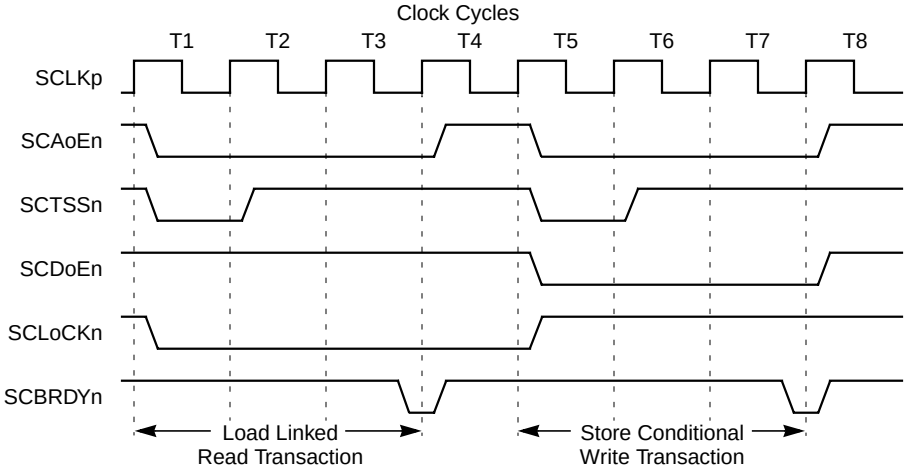
If an effective address for a Load Linked instruction is in the WriteBack cached area, the CW4011 does not assert SCLoCKn, even if it experiences a D-cache miss. The subsequent Store Conditional instruction does not generate a write transaction because it may hit the D-cache. If a Store Conditional instruction hits the D-cache in a WriteBack cached area when SCLoCKn is asserted, an incorrect condition occurs, and SCLoCKn is deasserted without any bus transactions being executed.

The effective virtual addresses of Load Linked and Store instructions must be in *kseg1*. Additionally, a Load Linked instruction and a Store Conditional instruction must be used as a pair of instructions to the same address.

While the CW4011 asserts SCLoCKn, the bus interface does not exhibit any special behavior—for example, it accepts hold requests. If a hold request is not accepted while the CW4011 is asserting SCLoCKn, outside user logic must mask the hold request by asserting SCLoCKn.

Figure 8.19 shows the timing behavior for locked transactions. If there are other transactions between the read transaction of a Load Linked and write transaction of a Store Conditional, the CW4011 asserts SCLoCKn continuously.

Figure 8.19 SCbus Locked Transaction



8.2.9 Big-Endian Configuration

The CW4011 can support big-endian address ordering, although the default configuration is little-endian. To enable big-endian mode, the BENDn input is strapped LOW. Table 8.3 lists the names arbitrarily used to describe the off-core address bus, data bus, and byte enable signals of the big-endian configuration. Since these signals are defined outside the CW4011 core, the actual names will be determined by the designer's choice of off-core logic.

Table 8.3 Big-Endian Arbitrary Signal Names

Signals	Big-Endian Signals
Address Bus	BiGE_Aip[31:0], BiGE_Aop[31:0]
Data Bus	BiGE_Dip[63:0][1], BiGE_Dop[63:0]
Byte Enables	BiGE_BEn[7:0]

1. BiGE_Dip[63] is the most-significant bit of a doubleword.

Table 8.4 lists the BiGE_BEn[7:0] bits and their corresponding BiGE_Dip and BiGE_Dop valid bits.

Table 8.4 Big-Endian Valid Bytes

BiGE_BEn[7:0] Bit	Byte Valid
0	BiGE_Dip[7:0] or BiGE_Dop[7:0]
1	BiGE_Dip[15:8] or BiGE_Dop[15:8]
2	BiGE_Dip[23:16] or BiGE_Dop[23:16]
3	BiGE_Dip[31:24] or BiGE_Dop[31:24]
4	BiGE_Dip[39:32] or BiGE_Dop[39:32]
5	BiGE_Dip[47:40] or BiGE_Dop[47:40]
6	BiGE_Dip[55:48] or BiGE_Dop[55:48]
7	BiGE_Dip[63:56] or BiGE_Dop[63:56]

These bit assignments are different from those of the SCDip[63:0], SCDop[63:0] and SCTBEn[7:0] signals. For big-endian mode, the data bus and byte enable signals outside the CW4011 need to be redefined, the most important of which is the definition of the byte enables. Table 8.5 shows the byte enable and data bus connections. The address bus bit assignments are not shown, but are direct connections from BiGE_Aip[31:0] to SCAip[31:0], and from BiGE_Aop[31:0] to SCAop[31:0].

Table 8.5 Data Bus and Byte Enable Connections

Signal	Big-Endian	Connection
Data Bus	BiGE_Dip[63:32]	SCDip[31:0]
	BiGE_Dip[31:0]	SCDip[63:32]
	BiGE_Dop[63:32]	SCDop[31:0]
	BiGE_Dop[31:0]	SCDop[63:32]
Byte Enables	BiGE_BEn[0]	SCTBEn[7]
	BiGE_BEn[1]	SCTBEn[6]
	BiGE_BEn[2]	SCTBEn[5]
	BiGE_BEn[3]	SCTBEn[4]
	BiGE_BEn[4]	SCTBEn[3]
	BiGE_BEn[5]	SCTBEn[2]
	BiGE_BEn[6]	SCTBEn[1]
	BiGE_BEn[7]	SCTBEn[0]

The above data bus configuration must be defined outside the CW4011 core. Table 8.6 shows different CW4011 data transactions through these buses.

Table 8.6 CW4011 Accesses through Off-Core Buses

Data Type	BiGE_Aop[2:0] Value	BiGE_BEn[7:0] Value	Valid Data	
			BiGE_Dop	BiGE_Dip
Byte	000	01111111	[63:56]	[63:56]
	001	10111111	[55:48]	[55:48]
	010	11011111	[47:40]	[47:40]
	011	11101111	[39:32]	[39:32]
	100	11110111	[31:24]	[31:24]
	101	11111011	[23:16]	[23:16]
	110	11111101	[15:8]	[15:8]
	111	11111110	[7:0]	[7:0]
Half-word	000	00111111	[63:48]	[63:48]
	010	11001111	[47:32]	[47:32]
	100	11110011	[31:16]	[31:16]
	110	11111100	[15:0]	[15:0]
Tribyte	000	00011111	[63:40]	[63:40]
	001	10001111	[55:32]	[55:32]
	100	11110001	[31:8]	[31:8]
	101	11111000	[23:0]	[23:0]
Word	000	00001111	[63:32]	[63:32]
	100	11110000	[31:0]	[31:0]
Doubleword	000	00000000	[63:0]	[63:0]

8.3 OCAbus Interface Behavior

The CW4011 on-chip access (OCA) bus enables access to on-chip modules at the CR stage without going through the SCbus. Section 7.4, “OCAbus Interface,” provides additional information about the bus. This section describes certain OCAbus transactions in detail and provides appropriate timing diagrams. These OCA transaction descriptions include:

- Basic OCA access
- Rejection of OCA access
- OCAbus access with stall at the EX pipeline stage
- OCAbus access with stall at the CR pipeline stage
- OCAbus access with stall request or wait state
- OCAbus access with pipeline cancellation

8.3.1 Basic OCAbus Transaction

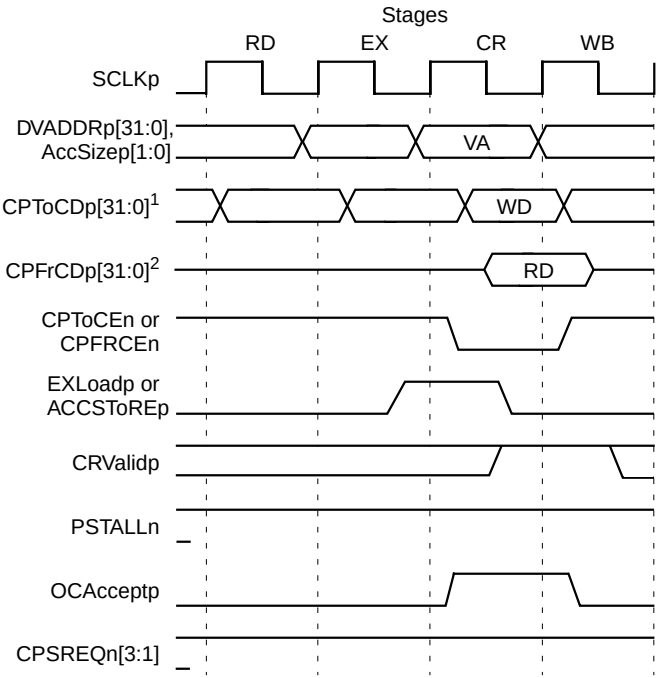
Regardless of the type of load or store execution, address and size are output at the EX stage of the CW4011 pipeline, and EXLoadp or ACCSToREp is asserted. The address bits (DVADDRp[31:0]) need to be decoded to determine whether or not OCAcceptp is asserted and the OCA module can accept the OCA transaction.

Typically, OCA module addresses should be located as uncached devices, so that the virtual address is in *kseg1*. This is done by setting address bits [31:29] to 0b101. The address bus must be latched on the rising edge of the system clock, between the EX and CR stages, as shown in Figure 8.20. The size information provided by AccSize is also latched at this time. Refer to the subsection entitled “AccSizep[1:0] OCAbus Transaction Size Output” on page 7-13 for more information on this subject.

At the CR stage, write data is output on CPToCDp provided that CPToCEn is asserted. If a read transaction is being executed, the CPFRCEn signal is asserted and data on the CPFRCDp bus is sampled on the rising edge of the system clock between stages CR and WB. The OCAcceptp signal must be asserted in the CR stage to inform the CW4011 that an OCA transaction is in progress.

The CRValidp signal is asserted to indicate that the CR stage is valid. If it is deasserted, write data must not be written and read data must not be sampled. The transaction is executed again later.

Figure 8.20 Typical OCAbus Transaction

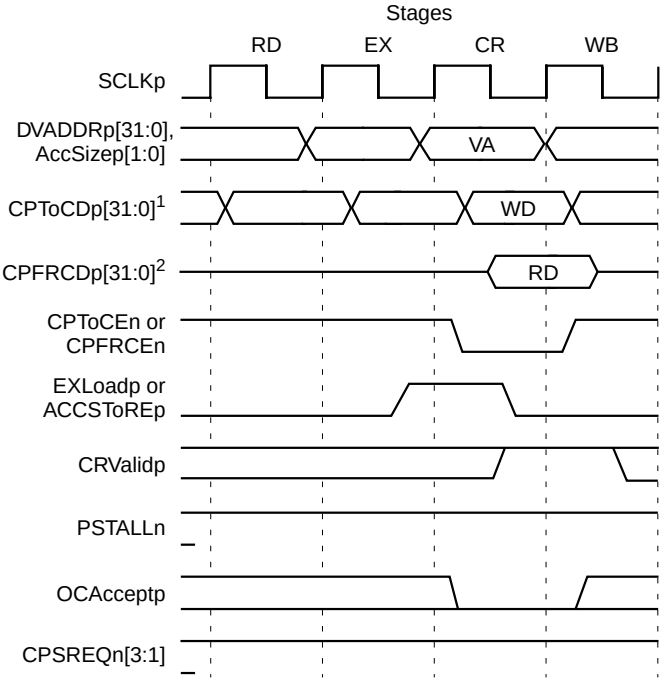


1. Write cycle.
2. Read cycle.

8.3.2 OCAbus Transaction Rejected

Figure 8.21 shows the timing for an OCAbus transaction that is rejected because OCAcceptp is deasserted during the CR stage. This occurs when the virtual address is decoded and found not to be an address for an OCA module. Under these conditions, the CW4011 reads from the D-cache, requests an SCbus read transaction, and then writes data to the D-cache write buffer or to a four-deep external write buffer. OCAcceptp is the only signal that determines whether an OCA transaction will take place.

Figure 8.21 OCAbus Transaction Rejected by Address Decoder

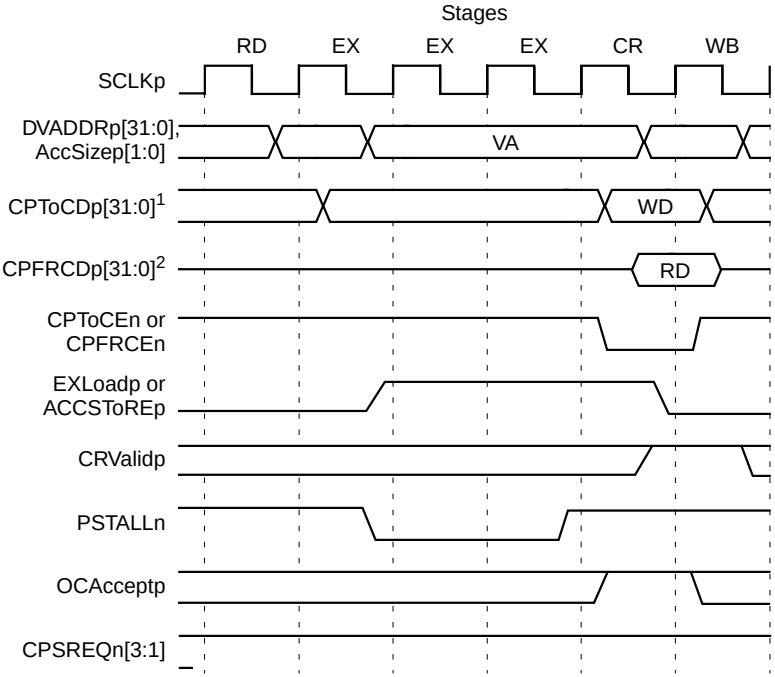


1. Write cycle.
2. Read cycle.

8.3.3 OCAbus Access with Stall at EX Stage

Figure 8.22 shows an example where PSTALLn is asserted at the EX stage of the CW4011 pipeline, causing all pipeline stages to enter a stall state. When this happens, DVADDRp[31:0], AccSizep[1:0], EXLoadp, or ACCSToREp are held during the stall cycles.

Figure 8.22 OCAbus with Stall at EX Stage

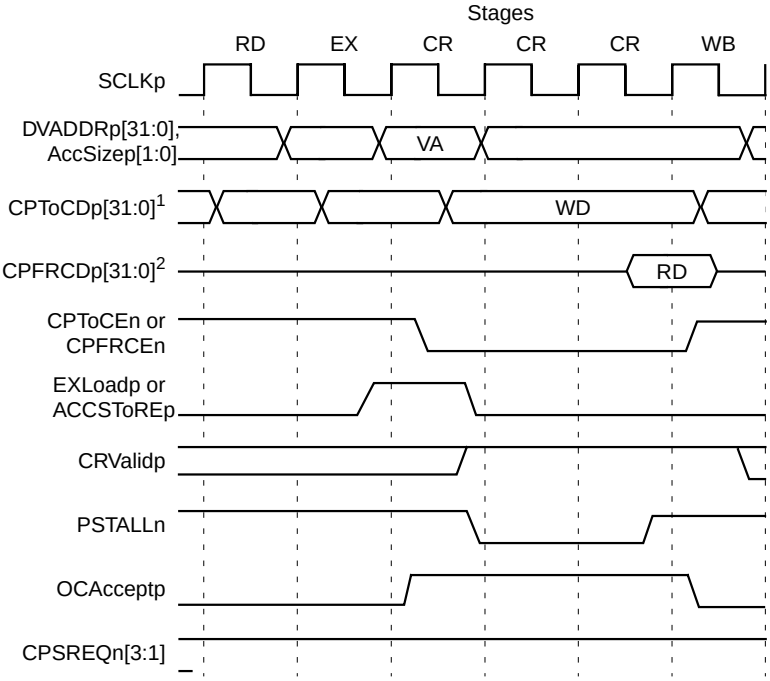


1. Write cycle.
2. Read cycle.

8.3.4 OCAbus Access with Stall at CR Stage

Figure 8.23 shows an example where PSTALLn is asserted at the CR stage of the CW4011 pipeline causing all pipeline stages to enter a stall state. When this happens, data on the CPToCDp bus, CRValidp, and OCAcceptp are held.

Figure 8.23 OCAbus Access with Stall at CR Stage

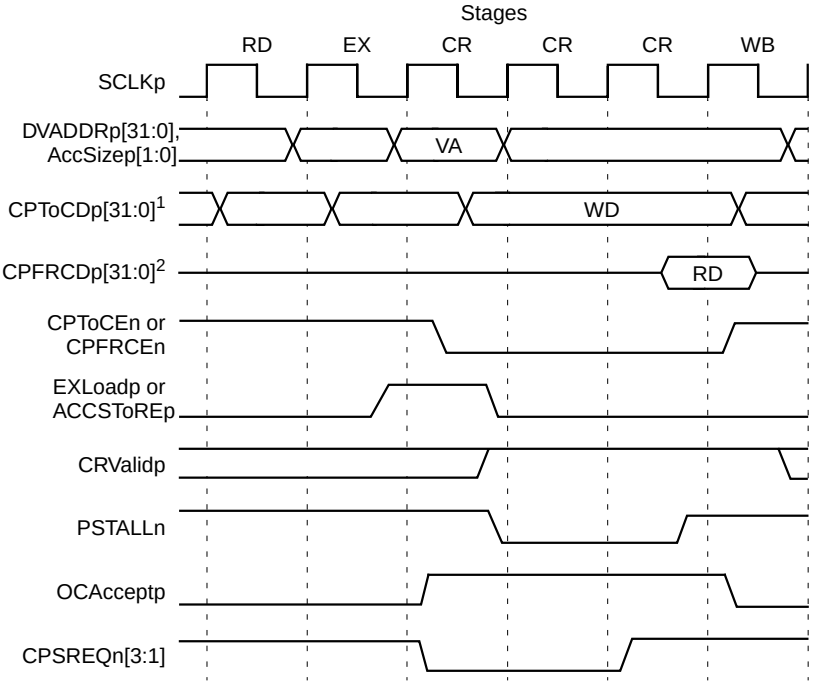


1. Write cycle.
2. Read cycle.

8.3.5 OCAbus Access with Stall Request

Figure 8.24 shows an example where the OCA bus device needs to insert some wait cycles before a read or write operation. To request a pipeline stall, the processor asserts CPSREQn from the beginning of the CR stage and this causes PSTALLn to be asserted. CPSREQn must be asserted and deasserted early in the clock cycle, since it is one of the critical path signals.

Figure 8.24 OCAbus Access with Stall Request

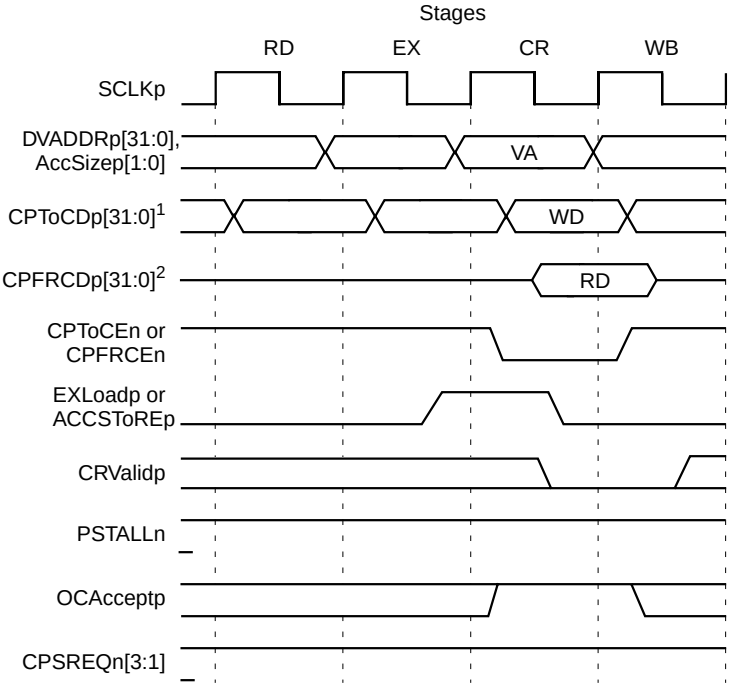


1. Write cycle.
2. Read cycle.

8.3.6 OCAbus Access with Pipeline Cancel

Figure 8.25 shows an example where a load or store instruction is cancelled by an exception. The exception is indicated when CRValidp is deasserted. When this happens, the write data must not be written into the OCA module. The read data being transferred to the CW4011 core is ignored. The cancelled load or store instruction may be executed later.

Figure 8.25 OCAbus Access with Pipeline Cancel



1. Write cycle.
2. Read cycle.

8.4 Cache Interface Behavior

When an external bus master writes data into main memory, it can invalidate the D-cache and I-cache lines to maintain coherency between the main memory and the caches. The CW4011 has three signals to support this function:

- Cache Invalidate Address Bus bits (CiNVAp[31:5])
- D-Cache Invalidate Strobe (DCiNVSn)
- I-Cache Invalidate Strobe (ICiNVSn)

When DCiNVSn or ICiNVSn is asserted, the address on the CiNVAp bus is latched and the CW4011 starts an invalidation process. DCiNVSn or ICiNVSn should be asserted for only one clock cycle. The D-cache or I-cache line is invalidated when the cache physical address tag, whose

line is valid, is coincident with the latched invalidate address. Both the V bit and the WB bit are cleared.

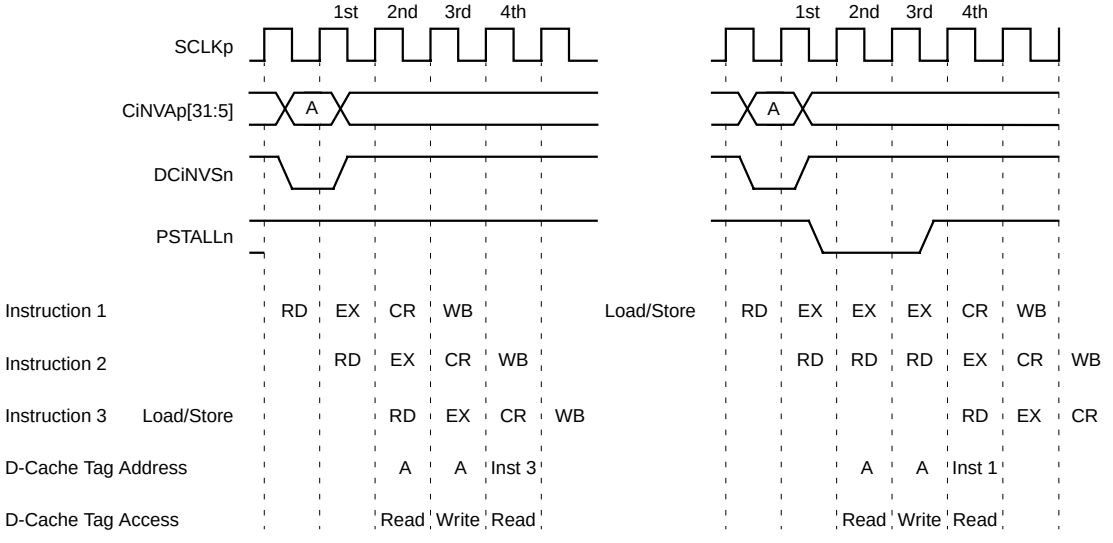
Figure 8.26 shows the timing diagram for D-cache invalidation implemented by bus snooping. In the first clock cycle after DCiNVSn is asserted, the LSU asserts the stall request signal if the EX stage is a load/store instruction.

In the second cycle, the D-cache tag is read from the D-cache and compared with the address of the latched DCiNVAp bus. If they match and the EX stage is a load/store instruction, the pipeline stall request is asserted. To avoid timing problems, PSTALLn may not be deasserted during the second cycle.

In the third cycle, the V bit and WB bit of the D-cache line are cleared and the line is invalidated. If the addresses do not match at the third cycle, the D-cache is not accessed.

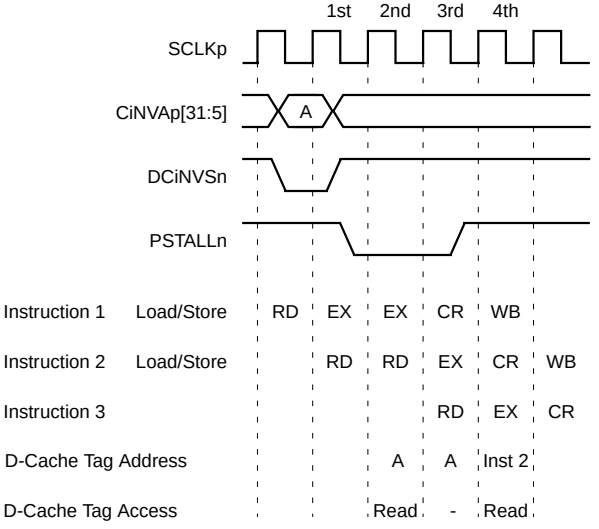
The stall cycle signal (PSTALLn) is asserted at the third clock cycle even if the address and D-cache tag do not match and the valid bit is not cleared.

Figure 8.26 D-Cache Invalidation by Snooping



Case 1. No Stall Cycle

Case 2. Two Stall Cycles

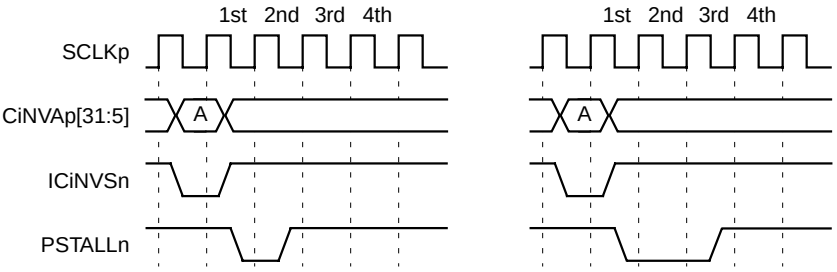


Case 3. One Stall Cycle (No Invalidation)

The LSU does not do anything if the external bus master read data from main memory and the address is dirty-cached by the CW4011. In this case, you may use WriteThrough mode for the page.

Figure 8.27 shows timing for I-cache invalidation brought about by bus snooping. It needs a two-cycle stall if the invalidation address hits the tag or a one-cycle stall if it does not hit the tag.

Figure 8.27 I-Cache Invalidation by Snooping



Case 1. One-Stall Cycle (Miss)

Case 2. Two-Stall Cycle (Hit)

Chapter 9
ICEport

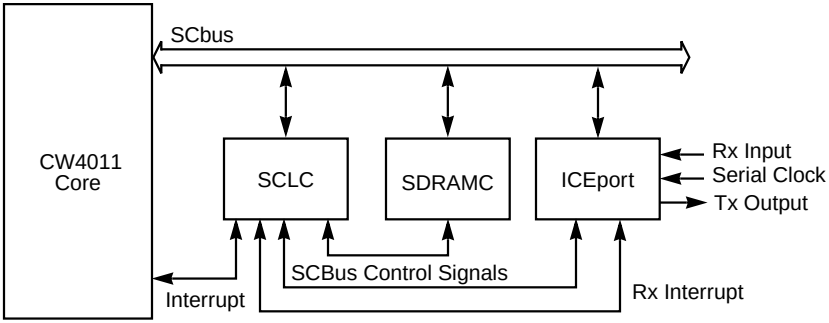
This chapter outlines the SerialICE scan interface and describes in detail the CW4011 ICEport building block. This chapter is divided into the following sections:

9.1 Overview

The ICEport is a full-duplex serial UART receive and transmit port building block available from LSI Logic. The core designer uses the ICEport both to download core application software and as a CW4011 debugging tool. The ICEport works with an ICEcontroller at baud rates up to 1 Mbit/s, providing 800 Kbits of data per second.

Figure 9.1 shows a block diagram of a CW4011 system with the ICEport installed. For LSI Logic’s LR4500 chip, the CW4011 ICEport is integrated with the SCLC and SDRAMC modules on the core SCbus. If desired, the ICEport can connect directly to the SCbus without the SCLC module.

Figure 9.1 CW4011 Design with ICEport



9.2 ICEport Features

The ICEport provides the following features:

- Full-duplex operation.
- Requires clock support at 16 times the transfer bit rate to define receiving (Rx) and transmitting (Tx) rates. This clock is common for Rx and Tx, and may be either an external clock or one generated internally from the system clock.
- Rx ready signal to indicate that a byte of data has been received and is in the data byte input buffer.
- Separate status and data registers for Rx and Tx. The Rx Status register contains one bit that indicates received data is in the ICEport, and one bit that indicates an overrun in the Rx input buffer. The Tx Status register contains one bit that indicates the ICEport is ready to transmit data.
- Serial-receive and clock input do not require an active signal when the ICEport is unused. During Reset, the Tx UART port defaults to an idle state and transmits an idle signal.

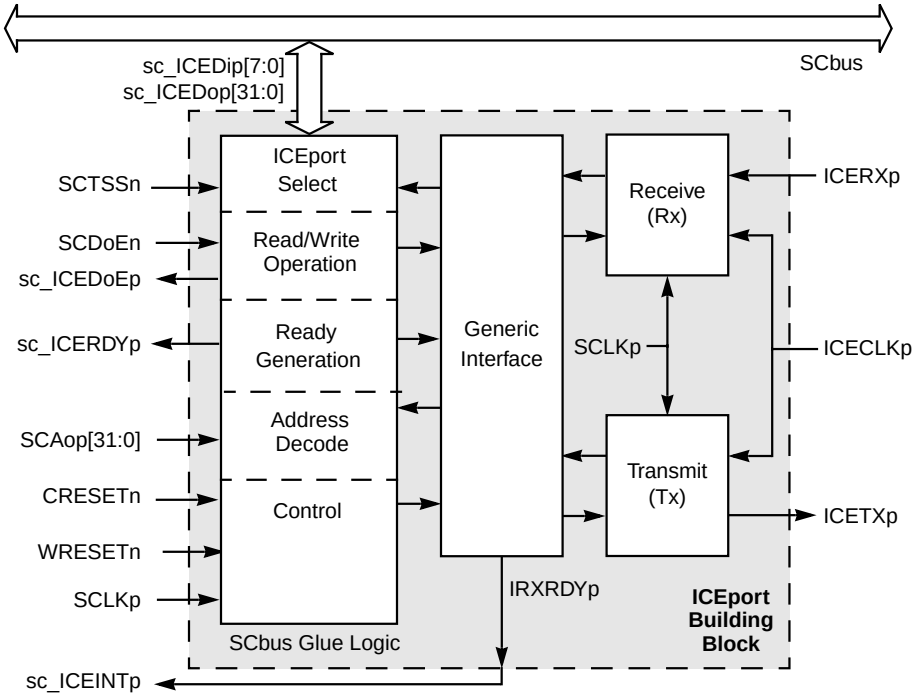
9.3 ICEport Functional Blocks

The CW4011 ICEport design has been partitioned into three logical blocks:

- Receive and Transmit Logic block, which sends and receives the ICETXp and ICERXp signals.
- Generic Interface Logic block, common to most core designs that implement a SerialICE ICEport.
- SCbus Interface Logic block, which connects the ICEport with the rest of the CW4011 core through the SCbus signals.

Figure 9.2 illustrates these blocks, their interactions with each other, and their interfaces to other cores and external logic.

Figure 9.2 CW4011 ICEport Block Diagram



9.3.1 Receive and Transmit Interface Logic

The two right-most blocks are the serial interface block, specifically the receive (Rx) and transmit (Tx) blocks. The Rx block receives the ICERXp bit stream, and the Tx block transmits the ICETXp bit stream. Both blocks receive the internal CPU clock (SCLKp) and the external x16 bit rate clock (ICECLKp). Both blocks synchronize timing between the ICECLKp and SCLKp timing domains. All interface signals between the Rx and Tx blocks and the Generic Interface are synchronized to SCLKp, since the Generic Logic block runs on SCLKp only.

9.3.2 Generic Interface Logic

The center Generic Interface block connects the Tx and Rx blocks to a specific core bus interface, which is the SCbus for the CW4011. The ICEport directly outputs only the IRXRDYp signal, which must be enabled in the Rx Setup register. When enabled, the IRXRDYp signal indicates that Rx data has been received. IRXRDYp is tied to the processor interrupt signal (sc_ICEINTp) and may be used for interrupt generation as described in Section 9.6.4.1, “Receive (Rx).”

9.3.3 SCbus Interface Logic

The left-most block in Figure 9.2 is the SCbus Interface Logic block. This block connects the Generic Interface to the CW4011 SCbus signals. The SCbus is the main internal CW4011 bus that allows a bus master to exchange information with the core. In SCbus transactions, the ICEport decodes the SCbus address line and checks the transaction start signal (SCTSSn) to see if the current SCbus transaction involves the ICEport. If the current transaction involves the ICEport, the SCbus Interface Logic block either places appropriate data on the data bus or writes data into an ICEport internal register, depending on whether the current operation is a read or a write. Once either transaction is complete, the ICEport asserts the acknowledge signal (sc_ICERDYp) and the SCbus Interface Logic block begins to monitor SCbus transactions again.

Please be aware that the ICEport follows a different SCbus protocol than other CW4011 core components. The ICEport uses only a certain subset of the entire SCbus signals and combines several SCbus acknowledge signals into a single ICEport signal. See Section 9.4.1, “Monitored SCbus Signals,” and Section 9.4.2, “Other SCbus Signals,” for more information on ICEport/SCbus interaction.

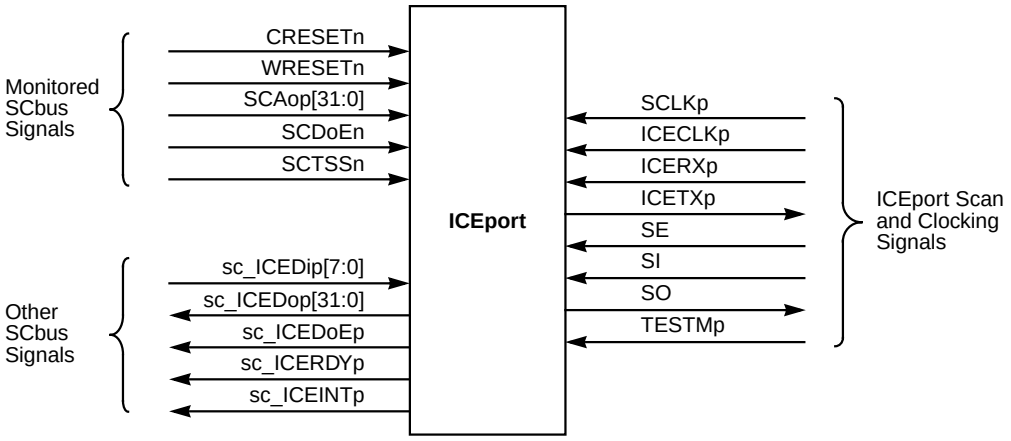
9.4 ICEport Signals

This section describes the signals that comprise the bit-level interface of the ICEport. The following paragraphs outline the conventions used in the signal descriptions:

- The signals are described in alphabetical order by mnemonic within each functional group. Each signal definition contains the mnemonic and the full signal name.
- The mnemonics for signals that are active HIGH, or for clock signals with a positive rising edge, end with a “p”; signals that are active LOW end with “n”.
- The term *assert* means to drive TRUE or active; *deassert* means to drive FALSE or inactive.
- *Input* and *Output* in the signal headings refer to I/Os with respect to the ICEport, not with the core. For example, SCTSSn is a core output, but because it is considered an ICEport input, it is labeled “Input.”
- All input signals, except for ICERXp and ICECLKp, are read on the positive edge of SCLKp and must therefore be generated synchronously with SCLKp.
- All output signals (except ICETXp) are also generated synchronously at the rising edge of the SCLKp clock. The ICETXp signal is synchronous to the rising edge of ICECLKp, except during a reset where ICETXp is asserted asynchronously to ICECLKp.
- In normal serial send and receive through the ICEport, ICECLKp runs at 16 times the rate of serial bit transmission/receive. This allows ICECLKp to define the bit width for each UART serial bit. The ICEport assumes that each serial bit for both receive and transmit is 16x ICECLKp, or 16 ICECLKp cycles.

Figure 9.2 summarizes the ICEport signals. Detailed descriptions follow the table. Note that the SCbus master can either be the SCLC module or the CW4011 processor. External logic refers to logic not related to the CW4011 core, the SCLC, or the ICEport.

Figure 9.3 ICEport Logic Diagram



9.4.1 Monitored SCbus Signals

This section lists the SCbus signals that the ICEport monitors and outlines how the ICEport uses these signals. For a more complete description of these signals, please see Chapter 7, "CW4011 Signals."

CRESETn **Cold Reset** **Input**
Asserting CRESETn asynchronously resets the ICEport and all ICEport registers. CRESETn and WRESETn are internally merged in the ICEport.

WRESETn **Warm Reset** **Input**
Asserting WRESETn asynchronously resets the ICEport and all ICEport registers. CRESETn and WRESETn are internally merged in the ICEport.

SCAop[31:0] **SCbus Address Bus** **Input**
SCAop[31:0] is the address bus. The ICEport monitors this bus and SCTSSn for data read/write operations involving the ICEport. When an SCbus transaction involves the ICEport, the ICEport decodes SCAop[31:0] to decide which internal register the transaction targets.

SCDoEn **SCbus Data Output Enable** **Input**
The value of SCDoEn determines whether the present SCbus transaction is a write or a read. If a write, SCDoEn is driven LOW; if a read, SCDoEn is driven HIGH. The

ICEport monitors SCDoEn so that it may perform the correct action for either a read or a write.

SCTSSn **SCbus Transaction Start Signal** **Input**
The core asserts SCTSSn for one clock cycle at the beginning of a transaction to announce that a new transaction has begun. Assertion of SCTSSn and a valid SCAop[31:2] address will initiate an ICEport read/write operation.

9.4.2 Other SCbus Signals

These signals enable ICEport read and write operations and transfer data for these operations.

sc_ICEDip[7:0]
SCbus Input Data Bus **Input**
This is the SCbus input data bus. For write operations to the ICEport, data transfers to the ICEport through this bus. On the same positive edge of SCLKp that asserts sc_ICERDYp, the core writes data into the ICEport.

sc_ICEDop[31:0]
SCbus Output Data Bus **Output**
This is the SCbus output data bus. For read operations from the ICEport, the ICEport will place data onto this bus. Data on this bus is valid for one clock cycle and only when the sc_ICEDoEp signal is asserted.

sc_ICEDoEp **SCbus Output Data Valid** **Output**
Asserting this signal indicates that the sc_ICEDop[31:0] bus is valid during the current cycle. sc_ICEDoEp asserts for read operations only and lasts only one SCLKp cycle.

sc_ICERDYp **ICEport Ready** **Output**
Asserting this signal HIGH informs the core or the SCLC module that the current transaction on the SCbus has finished. sc_ICERDYp encompasses both the SCB32n and SCBRDYn SCbus control signals.

sc_ICEINTp **ICEport Interrupt** **Output**
If this signal is enabled by the RxRXRDYPe bit in the Rx Setup register, the ICEport asserts sc_ICEINTp once it receives a valid byte of off-chip data. In the LR4500, the sc_ICEINTp output is sent to the SCLC module, which

then generates an interrupt to the core. sc_ICEINTp is also referred to as IRXRDYp in this document, since sc_ICEINTp is tied to the ICEport Generic Interface's IRXRDYp signal.

9.4.3 ICEport Scan and Clocking Signals

These signals are the clocking and scan I/O signals for the ICEport.

SCLKp **System Clock** **Input**
SCLKp is the global system clock input from the CW4011 core.

ICECLKp **ICE Serial Bit Clock Rate X16** **Input**
The ICEport requires that this off-chip signal have a clock frequency 16 times the serial transmit/receive rate. The ICEport assumes each serial/transmit bit is 16 ICECLKp cycles long.

ICERXp **Rx Serial Bit Receive** **Input**
This is an off-chip input that holds the UART serial input data stream. Each received bit is 16 ICECLKp cycles long.

ICETXp **Tx Serial Bit Transmit** **Output**
This is an off-chip output that holds the UART serial data stream. Each transmit bit is 16 ICECLKp cycles long.

SE **Scan Test Mode Enable** **Input**
Asserting this signal HIGH enables the scan chain and deasserting SE disables scan operation. The TESTMp signal must also be continuously asserted to enable the entire scan test.

SI **Scan Test Input** **Input**
SI is the scan chain data input signal.

SO **Scan Test Output** **Output**
SO is the scan chain data output signal.

TESTMp **Scan Test Setup** **Input**
This signal sets up the scan test, so that scan mode is possible in the SCLKp clock domain. TESTMp must be asserted continuously to enable the scan test.

The ICECLKp signal is ignored while TESTMp enables the scan test mode.

9.5 ICEport Registers

All ICEport registers are memory mapped as shown in Table 9.1. The default ICEport virtual base address is set to 0xB0FF0000 (0x10FF0000 physical address). Users can customize the ICEport address by altering the addresses in the HDL models. However, the last nibble (bits [3:0]) must be the kept the same, since these four bits determines which ICEport register to access. The addresses must also be both unmapped to prevent an installed MMU from remapping memory addresses and uncached to maintain data congruency. For these reasons, LSI Logic suggests using unmapped and uncached memory space *kseg1*.

Table 9.1 ICEport Registers

Register	Physical Address	Virtual Address	Reference Page
Rx Status	0x10FF0000	0xB0FF0000	9-10
Rx Setup	0x10FF0000	0xB0FF0000	9-11
Rx Data	0x10FF0004	0xB0FF0004	9-11
Tx Status	0x10FF0008	0xB0FF0008	9-12
Tx Data	0x10FF000C	0xB0FF000C	9-12

All register read transactions return zeroes for bits [31:8], and data for bits [7:0]. For read operations, the register bits are mapped with SCDip[31] to sc_ICEDop[31], and so on. For write operations, the register bits are mapped with SCDop[7] to sc_ICEDip[7], and so on. During write operations, data on SCDop[31:8] is ignored, write transactions to read-only registers are ignored, and read transactions from the write-only registers return undefined data.

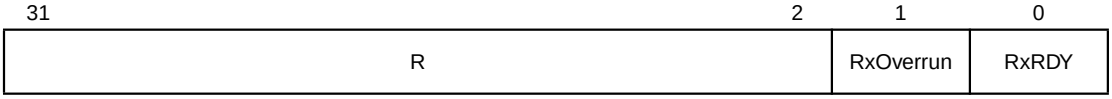
All registers must be accessed only using word accesses to avoid conflict between big-endian and little-endian data structures, and to avoid partial update problems.

Note that each bit field within a register is described by mnemonic in the register figure, and by mnemonic, full name, bit number, and read/write status within the following bit field description.

9.5.1 Rx Status Register

The read-only Rx Status register provides status information for ICEport receive operations and indicates the state of the Rx Data register. Figure 9.4 shows the Rx Status register.

Figure 9.4 Rx Status Register



R **Reserved** **[31:2]**
These bits are reserved for use by LSI Logic and are read as zeroes.

RxOverrun **Rx Overrun** **1**
This bit is set to one when an Rx overrun error occurs. An Rx overrun error occurs when a new Rx byte is received, as indicated by RxRDY, before the previous Rx byte has been read. For an overrun error, the new byte is not accepted and the pending byte in the Rx Data register is not lost.

When the RxOverrun bit is set, it signals that at least one byte from the serial input stream of the new frame has been lost. RxOverrun is cleared when the Rx Status register is read. This ensures that if another overrun occurs between the Rx Status register read and the Rx Data register read that this overrun will set RxOverrun.

RxOverrun clears to zero during an ICEport reset.

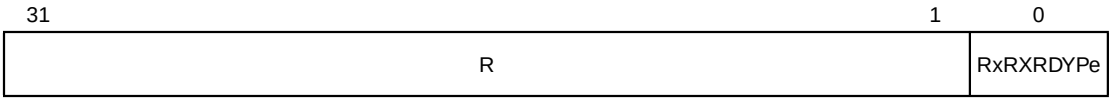
RxRDY **Rx Byte Ready** **0**
When the Rx block receives a byte, this bit is set to one. RxRDY clears to zero when the Rx Data register is read, and at reset. The IRXRDYp (sc_ICEINTp) output signal, if enabled, reflects the state of the RxRDY bit.

9.5.2 Rx Setup Register

The write-only Rx Setup register enables and disables the sc_ICEINTp interrupt signal when the RxRDY bit in the Rx Status register is set. If software clears the RxRXRDYPe bit to zero, then the ICEport interrupt signal sc_ICEINTp is disabled. This feature was added to allow software to disable the interrupt signal, IRXRDYp (sc_ICEINTp), if the IRXRDYp signal were tied to a Not Maskable Interrupt (NMI) input.

Figure 9.5 shows the Rx Setup register, with bit field descriptions following the figure.

Figure 9.5 Rx Setup Register



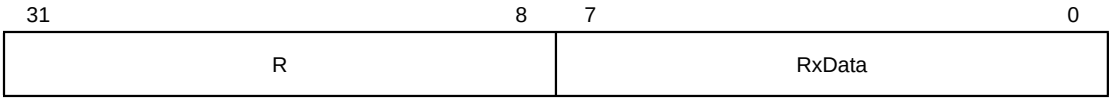
R **Reserved** **[31:1]**
These bits are reserved for LSI Logic, and any writes to these bits are ignored.

RxRXRDYPe **sc_ICEINTp(IRXRDYp) Enable** **0**
When this bit is set to one, the sc_ICEINTp signal reflects the state of the RxRDY bit in the Rx Status register. When software clears RxRXRDYPe to zero, the sc_ICEINTp signal is continually deasserted. RXRXRDYPe clears to zero during an ICEport reset.

9.5.3 Rx Data Register

The read-only Rx Data register, shown in Figure 9.6, holds received data in bits [7:0]. RxData is valid only when the RxRDY bit in the Rx Status register is set. The Rx Data register is undefined after an ICEport reset.

Figure 9.6 Rx Data Register



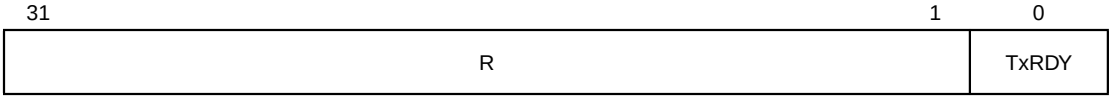
R **Reserved** **[31:8]**
These bits are reserved for LSI Logic and are read as zeroes.

RxData **Received Bit Stream** **[7:0]**
This bit field holds data received from the ICERXp serial input signal. Data held in RxData is valid only when the RxRDY bit in the Rx Status register is set. RxData is undefined after an ICEport reset.

9.5.4 Tx Status Register

The read-only Tx Status register, shown in Figure 9.7, provides status information for Tx operations.

Figure 9.7 Tx Status Register



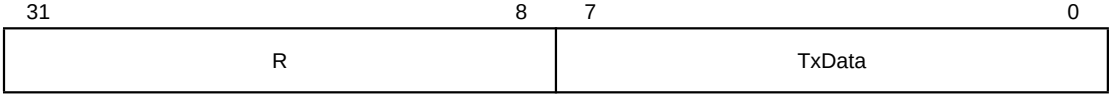
R **Reserved** **[31:1]**
These bits are reserved for LSI Logic and are read as zeroes.

TxRDY **TxRDY Rx Ready** **0**
This bit is set to one when either the Tx Data register is ready for the next transmit byte, or after reset. TxRDY remains set during and after the transmission of Tx data. TxRDY clears to zero during a write to the Tx Data register. TxRDY is set to one after an ICEport reset.

9.5.5 Tx Data Register

The write-only Tx Data register, shown in Figure 9.8, holds the serial transmission data.

Figure 9.8 Tx Data Register



R **Reserved** **[31:8]**
These bits are reserved for LSI Logic, and any writes to these bits are ignored.

TxData	**Transmitted Bit Stream** When the TxRDY bit in the Tx Status register is set, data for transmit through ICETXP may be written to the TxData bits. Writes to the TxData bits when the TxRDY is zero are ignored.	**[7:0]**

9.6 ICEport Operations

This section describes the different operations of the ICEport, and is divided into the following sections:

- Section 9.6.1, “SCbus Read/Write Transactions”
- Section 9.6.2, “Reset”
- Section 9.6.3, “Serial Bit Stream”
- Section 9.6.4, “ICEport Receive and Transmit”
- Section 9.6.5, “Clock Domains and Properties”

9.6.1 SCbus Read/Write Transactions

All read or write operations to the ICEport occur through the SCbus. Both transactions require two cycles once SCbus arbitration is decided. For either transaction, the bus master first must win arbitration for SCbus control and decide to initiate a transaction. The bus master then places the target address for the transaction on SCAop[31:0] and asserts SCTSSn for one cycle to indicate the start of a new operation. The ICEport constantly decodes SCAop[31:0] and monitors SCTSSn for transactions that target the ICEport. If the ICEport is the target of an operation, it checks the SCDoEn signal to determine whether this transaction is a read or a write.

For a read, the ICEport places data on sc_ICEDop output bus, asserts sc_ICERDYp, and then asserts sc_ICEDoEp at the next clock cycle.

For a write, the ICEport latches data on sc_ICEDip into the proper register on the next rising edge of the clock and asserts sc_ICERDYp at the following clock cycle. In order to ensure that information is not lost, the SCbus master must hold the SCAop[31:0], SCDoEn, and SCDop[31:0] signals until the ICEport asserts the sc_ICERDYp acknowledge signal.

For data transfer, the SCDop[7:0] output bus connects to the ICEport sc_ICEDip[31:0] input bus. The SCDip[31:0] input bus connects to the ICEport sc_ICEDop[31:0] output bus. The upper 32 bits of both SCbus data buses SCDop[63:32] and SCDip[63:32] are not used for ICEport transactions.

Figure 9.9 shows an ICEport read, and Figure 9.10 shows an ICEport write. For both figures, the signals CRESETn, WRESETn, SE, and TESTMp are assumed deasserted throughout the transaction. All read/write operations are synchronous to the rising edge of the SCLKp. Detailed descriptions follow the figures.

Figure 9.9 Read Operation

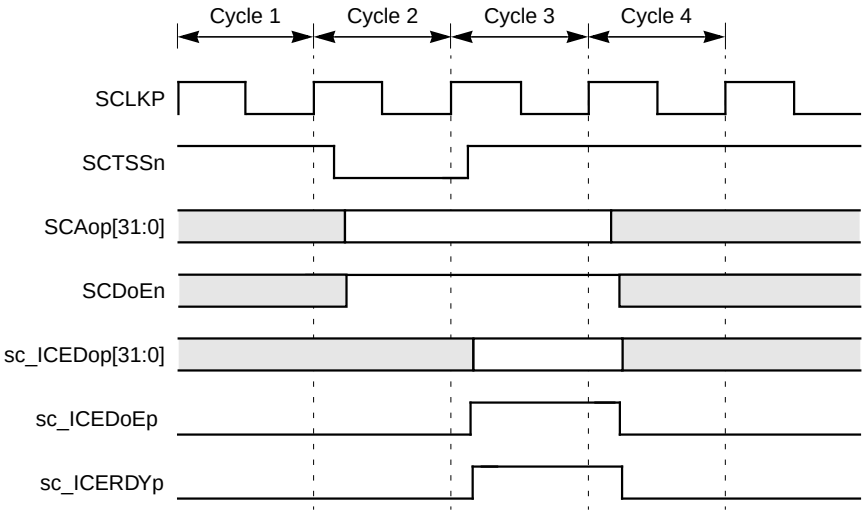
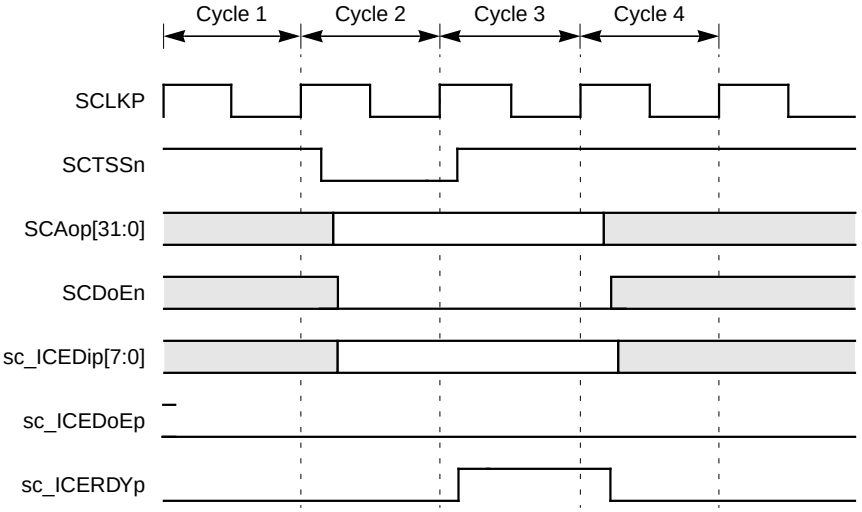


Figure 9.10 Write Operation



The following comments outline operations during cycles 1 to 4 presented in both figures.

Cycle 1: The bus master wins arbitration of the SCbus.

Cycle 2: The bus master asserts SCTSSn for one cycle to indicate the start of a new transaction. It also places the target address on SCAop[31:0] and asserts SCDoEn for a write operation, or deasserts SCDoEn for a read operation. For a write, the bus master also drives SCDop[31:0] with the data to be transferred.

Cycle 3: The ICEport recognizes that it is the transaction target. For a read, the ICEport places the appropriate data on the sc_ICEDop[31:0] bus and asserts sc_ICEDoEp. For a write, the ICEport writes sc_ICEDip[7:0] data into the appropriate register. The ICEport then asserts sc_ICERDYp to indicate that the transaction has finished.

Cycle 4: The ICEport deasserts sc_ICERDYp at the rising edge of SCLKp. For a read transaction, the ICEport also deasserts sc_ICEDoEp and the SCbus master must latch the data on the rising edge of SCLKp at the start of this cycle. At the end of Cycle 4, the ICEport is ready to begin a new transaction.

9.6.2 Reset

An ICEport system reset occurs when either CRESETn or WRESETn is asserted for at least one SCLKp cycle. CRESETn must be asserted when the system is powered up in order to set the ICEport in a predefined state. Since the reset signals are synchronous to SCLKp, the ICEport can be reset even if the ICECLKp clock is not running.

An ICEport system reset performs the following functions:

- RxOverrun and RxRDY bits in the Rx Status register are cleared, indicating that the Rx Data register is undefined.
- The RxRXRDYPe bit in the Rx Setup register is cleared. This causes the IRXRDYp (sc_ICEINTp) signal to be deasserted.
- The TxRDY bit in the Tx Status register is set.

9.6.3 Serial Bit Stream

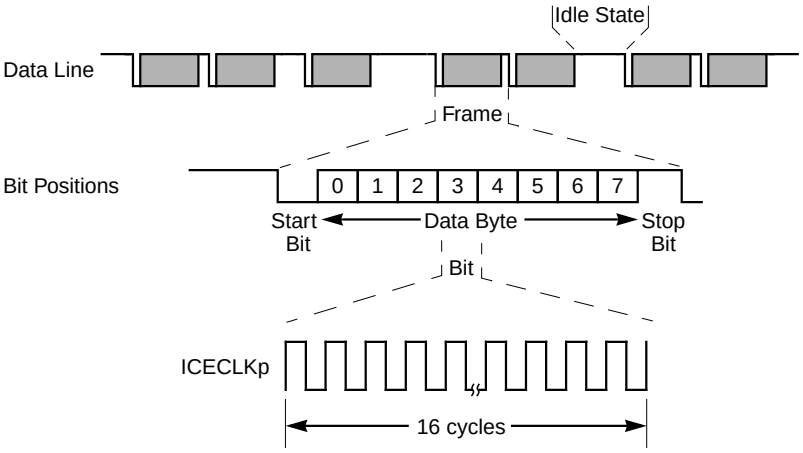
The ICEport receives data on ICERXp and transmits it on ICETXp in serial bit streams. In the Receive (Rx) block, the ICEport receives data. When no data is being transferred, the Transmit (Tx) block holds ICETXp idle HIGH.

Figure 9.11 shows an interpretation of the serial bit stream on the data line. The data bytes are received in frames, with each frame consisting of three pieces:

- a start bit, always LOW
- a byte of data, transmitted true level from LSB (bit 0) to MSB (bit 7)
- and a stop bit, always HIGH

All bits in a frame are 16 ICECLKp cycles long. The data line remains HIGH after the stop bit when the line goes idle, until the next start bit drives the line LOW.

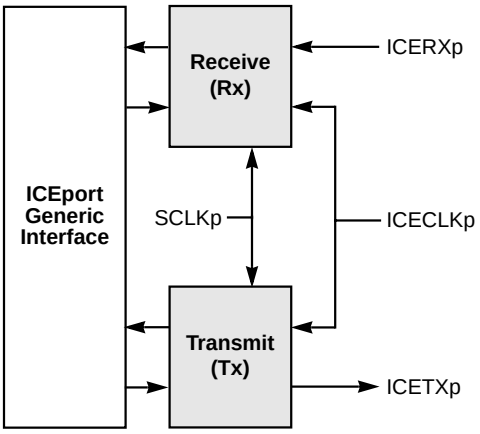
Figure 9.11 Serial Bit Stream



9.6.4 ICEport Receive and Transmit

There are two ICEport serial interface blocks, specifically the receive (Rx) and transmit (Tx) blocks. The Rx block receives the ICERXp bit stream, and the Tx block transmits the ICETXp bit stream. Both blocks receive the internal CPU clock (SCLKp) and the external bit rate clock (ICECLKp). Both blocks synchronize timing between the ICECLKp and SCLKp timing domains. Figure 9.12 shows a simple block diagram of the Rx and Tx blocks (shaded) with signals and clocking.

Figure 9.12 Rx and Tx Blocks



The remainder of this section details both the receive and transmit ICEport operations.

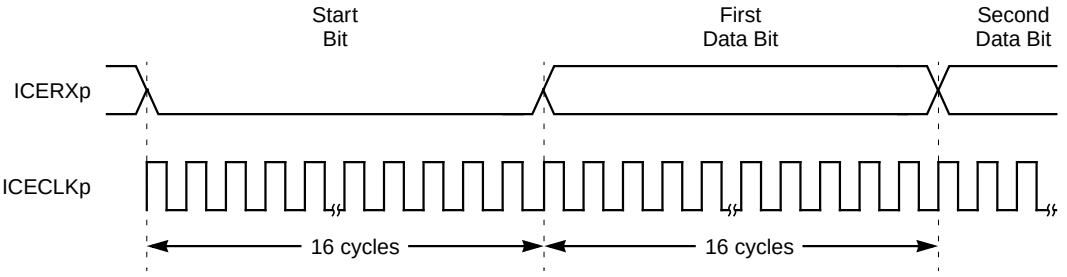
9.6.4.1 Receive (Rx)

ICERXp is the serial data input to the ICEport. The Rx block receives the ICERXp signal and reads it on the rising edge of ICECLKp. ICECLKp can be used to generate both the transmit and receive data clocks, but usually two different clocks are implemented. This is not a problem, as long as the difference between the two clock frequencies is below a certain limit, as outlined in Section 9.6.5, “Clock Domains and Properties.”

The Rx block is synchronized when ICERXp has been HIGH for nine bit times (144 ICECLKp cycles) or more, which indicates that the data line is in an idle state. The Rx block must be synchronized after power on, reset, serial cable connection, or any other event that would alter Rx block synchronization.

After synchronization, the Rx block begins sampling ICERXp on each rising edge of ICECLKp. When it samples a LOW ICERXp value, the Rx block recognizes this as the start bit of a new data frame and prepares for the serial data stream. The width of each received bit is assumed to be 16 ICECLKp cycles, even though the clock that generated the data for ICERXp may be different from the ICECLKp. The value of ICERXp at the eighth ICECLKp rising edge is assumed to be the value of the bit, and the bit is then received. If the start bit is HIGH, the frame is ignored. In this case, the ICERXp LOW value that indicated the start of the frame was accidental. Figure 9.13 shows the serial bit clocking relative to ICECLKp.

Figure 9.13 Received Bit Timing



The Rx block places the eight data bits received after the start bit in the Rx Data register. The first data bit received after the start bit is the LSB (bit 0), and the eighth data bit received is the MSB (bit 7). The eight data bits received between the start and stop bits are all true level values.

A valid HIGH stop bit received at the end of the frame sets the RxRDY bit in the Rx Status register. The IRXRDYp (sc_ICEINTp) output reflects the state of the RxRDY bit, if enabled by the RxRXRDYPe bit in the Rx Setup register. IRXRDYp can be used as an interrupt to ensure that the CPU reads the data received, thus avoiding overruns. If the stop bit is LOW, the frame is ignored.

A received data byte is not placed in the Rx Data register until a valid stop bit is received. This data byte will be available throughout the next data byte (frame) receive, until the next valid stop bit refreshes the Rx Data register. In other words, a previously received data byte is present in the Rx Data register for at least nine bit cycles (144 ICECLKp cycles) after a new start bit (for a new frame) is received.

If a previously received byte has not been read when a new byte is ready for the Rx Data register, an overrun error occurs. When an overrun error occurs, the ICEport sets the RxOverrun bit in the Rx Status register, and the new frame is discarded.

If the ICEport receives an invalid stop bit, the stop bit is not recorded by the ICEport registers and the frame is still discarded. The ICEport will not accept a new start bit until the previous frame has been finished by a valid stop bit or a HIGH value on ICERXp. This ensures that the ICEport will not indicate a runaway receive if ICERXp is tied LOW in error. Therefore, the ICEport will not receive a frame after reset if ICERXp is continuously either HIGH or LOW.

When the Rx block receives the stop bit correctly, a LOW value in the bit stream immediately following the stop bit will start the next frame. The start bit must be allowed to begin this quickly, since ICECLKp may be slower than the clock that generates the data for ICERXp. In such a case, the next received frame may start on the next sample ICECLKp.

9.6.4.2 Transmit (Tx)

The ICETXp signal is the ICEport serial data output and can carry new data every 16 ICECLKp cycles. When there is no data for transmission,

ICETXp is held HIGH in an idle state. During this idle state, the TxRDY bit in the Tx Status register is set to one, which indicates that transmission may be initiated by placing data in the Tx Data register. After data is written to the Tx Data register, the ICEport clears the TxRDY bit to zero.

Start bit transmission begins on the rising edge of ICECLKp and the first data bit starts transmitting 16 ICECLKp clock cycles later. Every bit of the transmitted frame has a width of 16 ICECLKp cycles. The Tx Data register LSB (bit 0) is transmitted just after the start bit; the MSB (bit 7) is sent just before the stop bit. All data bits are transmitted true level, with zeroes sent as LOW values and ones sent as HIGH values.

The ICEport sets the TxRDY bit in the Tx Status register when data bit 7 (the end of the byte) begins transmitting. As soon as TxRDY is set, the next data byte to transmit can be written to the Tx Data register. Writing to the Tx Data register while either data bit 7 or the stop bit is transmitting ensures that the ICETXp signal will not be idle. If the next data byte is not written to the Tx Data register before the stop bit is transmitted, the Tx block will idle for a number of ICECLKp cycles, until new data is available in the Tx Data register.

9.6.5 Clock Domains and Properties

Since data commonly moves between the ICECLKp domain and the Rx clock domain, these two clocks must have frequencies within certain limits. The difference between the ICECLKp frequency and the ICERXp clock frequency may be no more than ±1%, with ICERXp jitter margins ±10% of the bit width. This jitter can originate from transmission cables or different timing in LOW-to-HIGH and HIGH-to-LOW transitions.

The UART receiving the output from ICETXp may, however, require less difference between the two frequencies, and this requirement must be observed.

The ICECLKp signal may be derived from SCLKp by using a divider. This method frees a pin since ICECLKp no longer requires an external pin. The operation of the ICEport does not change in any way if ICECLKp is derived from SCLKp, but the frequency difference of ±1% must be adhered to regardless of the clock rate.

The ICEport may also transfer data internally between the two clock domains (between ICEport and core). For these transactions, the ICECLKp frequency can be at most one fourth of the SCLKp frequency. No matter what the frequency difference between ICECLKp and SCLKp, the bus master must have enough time to read received data before new data arrives or an overrun error will occur.

9.7 ICEport Pin Buffers and Drivers

The choice of ICEport external pin buffers and drivers will vary with each design. However, this section provides a few general recommendations for any design using an ICEport. Please note that the pin reserved for ICECLKp may be conserved if the ICEport clock is internally derived from SCLKp, as described in Section 9.6.5, “Clock Domains and Properties.”

- The buffer for input pin ICERXp should be a 5-V-–compatible Schmitt trigger with an internal pull-up resistor, since the incoming signal may be noisy and driven from a 5-V source. An internal pull-up resistor is recommended so that ICERXp can be left unconnected if the ICEport is unused.
- The driver for the ICETXp output pin should be a 4-mA driver, with a reduced slew rate to avoid reflections.

Chapter 10
Specifications

This chapter specifies the physical and electrical characteristics of the CW4011 core. It contains the following sections:

10.1 Physical Specifications

The CW4011 has a single 1x clock input. Clock duty cycle may vary from 40 to 60% at maximum frequency. The CW4011 operates at 90 MHz for worst case process, 3.14 V, 110 ˚C at junction. The CW4011 dissipates approximately 7.0 mW/MHz. Table 10.1 lists the dimensions of the CW4011 core in G10 technology.

Table 10.1 CW4011 Physical Layout Size

Core	Technology	Width	Height	Total Area
CW4011	G10-p	2.5 mm	3.5 mm	8.75 mm^2

10.2 AC Timing and Loading

The input *setup* time is defined from the signal valid to the rising edge of SCLKp and the input *hold* time is defined from the rising edge of SCLKp to the signal valid. For input setup times, the driver must drive the signal valid before any receivers need it. For input hold times, the driver must hold the signal valid longer than needed by any receiver.

The output *maximum* and *minimum delay* times are defined from the rising edge of SCLKp to the signal valid.

Load is the total load on the net in standard loads visible to the output driver. The loading values are for internal loading only. The load column shows the internal loading on each net in the module.

Table 10.2 shows the timing conditions.

Table 10.2 CW4011 Timing Considerations

AC Timing	Process	V_{DD} (Volts)	Junction Temperature (˚C)	Clock Period (ns)
BCCOM	0.874	3.46	0	11.0
WC110	1.38	3.14	110	11.0

Figure 10.1 shows how the AC timing is defined. Table 10.3 and Table 10.4 list the AC timing values and the loading for the CW4011. The timing is from Motive Static Timing Analysis.

Note: The conditions used in this timing analysis were chosen to be representative of a typical CW4011 design and are intended as a guide to designers. However, the numbers obtained for individual designs may vary since loading depends on chip placement and routing. If the loading exceeds the values given on the previous page, the timing values may exceed those listed here.

Figure 10.1 AC Specifications

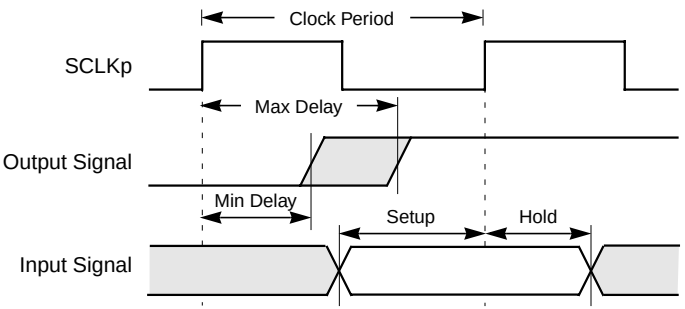


Table 10.3 CW4011 Input AC Timing and Loading

Signal Name	BCCOM		WC110		Standard Loads (pF)
	Setup (ns)	Hold (ns)	Setup (ns)	Hold (ns)	
BENDn[1]	—	—	—	—	0.75
CiNVAp[31:5]	–0.23	0.59	–0.39	1.12	0.75
CPBUSYn[3:1]	3.03	–0.56	5.72	–1.05	0.75
CPCoNDp[3:0]	2.81	–0.54	5.14	–1.04	0.75
CPFRCDp[31:0]	1.30	–0.05	2.51	–0.02	0.75
CPSREQn[3:1]	2.86	–0.79	5.13	–1.41	0.75
CRESETn	4.11	–0.31	7.11	-0.55	0.75
DCiNVSn	0.11	0.24	0.25	0.45	0.75
EXiNTn[5:0]	0.22	0.10	0.37	0.19	0.75
EXVAp[31:2]	1.82	–0.39	3.65	–0.90	0.75
EXViNTn	0.16	–0.05	0.28	–0.10	0.75
FPEoDDn	–0.07	0.17	–0.11	0.25	0.75
FPERRXn	0.06	0	0.18	–0.02	0.75
ICiNVSn	0.11	0.24	0.25	0.45	0.75
iMPLop[3:0]	1.58	–0.87	3.19	–1.80	0.75
NMin	0.34	0.21	0.65	0.41	0.75
OCAcceptp	1.98	–0.13	3.75	–0.26	0.75
REVLop[3:0]	1.64	–0.70	3.32	–1.46	0.75
SCB32n	2.66	–0.16	4.80	–0.22	0.75
SCBERRn	2.71	–0.29	4.91	–0.55	0.75
SCBPWAn	0.64	–0.34	1.22	–0.69	0.75
SCBRDYn	2.66	–0.36	4.80	–0.64	0.75

(Sheet 1 of 2)

Table 10.3 CW4011 Input AC Timing and Loading (Cont.)

Signal Name	BCCOM		WC110		Standard Loads (pF)
	Setup (ns)	Hold (ns)	Setup (ns)	Hold (ns)	
SCBRTYn	2.60	–0.33	4.71	–0.55	0.75
SCDip[63:0]	0.28	0.33	0.62	0.60	0.75
SCHRQn	3.01	-0.37	5.0.75	–0.70	0.75
SCTSEn	2.70	–0.27	4.51	–0.56	0.75
TESTMp1	—	—	—	—	0.75
WRESETn	4.08	–0.11	7.06	–0.25	0.75
(Sheet 2 of 2)					

1. BENDn and TESTMp are strapped input signals and do not change state.

Table 10.4 CW4011 Output AC Timing and Loading

Signal Name	BCCOM		WC110		Standard Loads (pF)
	MIn (ns)	Max (ns)	Min (ns)	Max (ns)	
AccSizep[1:0]	0.87	0.89	1.64	1.68	0.75
ACCSToREp	0.94	0.98	1.76	1.86	0.75
BRLiKFn	0.82	2.74	1.57	5.17	0.75
CPCoDEp[31:0]	1.16	3.70	2.28	6.74	0.75
CPFiXUPn	1.06	1.62	1.96	3.07	0.75
CPFRCEn	1.57	4.27	2.93	7.75	0.75
CPMISSn	1.66	4.11	2.98	7.50	0.75
CPRSTn[3:1]	0.78	0.79	1.46	1.49	0.75
CPToCDp[31:0]	0.94	3.53	1.74	6.87	0.75
CPToCEn	0.98	3.88	1.89	7.13	0.75
CPXoDDn	1.72	2.85	3.24	5.24	0.75
(Sheet 1 of 3)					

Table 10.4 CW4011 Output AC Timing and Loading (Cont.)

Signal Name	BCCOM		WC110		Standard Loads (pF)
	MIn (ns)	Max (ns)	Min (ns)	Max (ns)	
CPXSTBn[3:0]	1.31	4.81	2.41	8.69	0.75
CRValidp	1.03	3.38	1.93	6.12	0.75
DVADDRp[31:0]	1.16	2.92	2.07	5.55	0.75
EXLoadp	0.85	0.85	1.60	1.61	0.75
EXVApEn	0.84	0.85	1.60	1.61	0.75
PCANCRn	1.32	2.88	2.43	5.14	0.75
PCANoDDn	1.08	3.19	1.92	5.75	0.75
PSTALLn	1.46	3.76	2.65	6.93	0.75
SCANREQp	0.94	3.64	1.72	6.76	0.75
SCAoEn	0.66	0.76	1.34	1.39	0.75
SCAop[31:0]	0.70	0.82	1.39	1.60	0.75
SCBGEp	0.72	0.79	1.31	1.48	0.75
SCDoEn	0.80	0.96	1.59	1.70	0.75
SCDop[63:0]	0.71	0.80	1.39	1.57	0.75
SCHGTn	0.80	0.85	1.53	1.59	0.75
SCiFETn	0.57	0.69	1.14	1.41	0.75
SCLoCKn	0.58	0.64	1.17	1.31	0.75
SCTBEn[7:0]	0.62	0.74	1.22	1.49	0.75

(Sheet 2 of 3)

Table 10.4 CW4011 Output AC Timing and Loading (Cont.)

Signal Name	BCCOM		WC110		Standard Loads (pF)
	MIn (ns)	Max (ns)	Min (ns)	Max (ns)	
SCTBLn	1.04	1.23	2.00	2.22	0.75
SCTBSTn	0.57	0.69	1.14	1.41	0.75
SCTPWn	0.57	0.69	1.14	1.41	0.75
SCTRQn	1.23	3.12	2.31	5.85	0.75
SCTSSn	0.57	0.69	1.14	1.41	0.75
SUSPEXn	1.03	1.68	1.91	3.20	0.75
(Sheet 3 of 3)					

Appendix A
CW4011 Register Summary

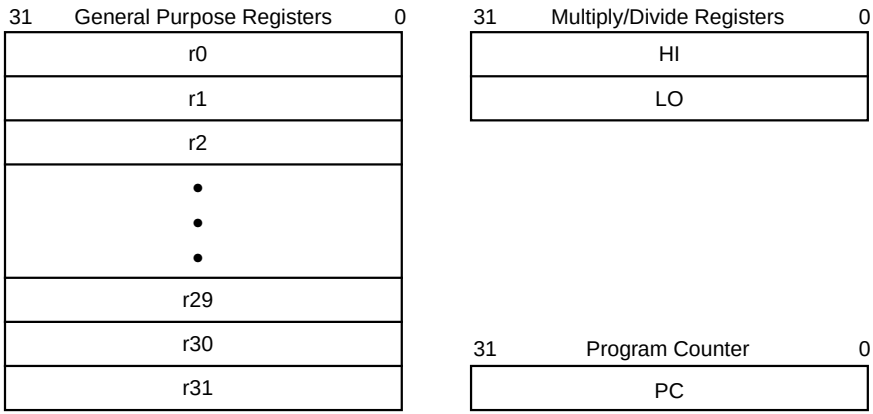
This appendix contains a quick description of all the CW4011 core registers and a listing of all CW4011-specific registers. This appendix is divided into two sections:

- Section A.1, “CW4011 CPU Registers”
- Section A.2, “Register Summary”

A.1 CW4011 CPU Registers

Figure A.1 shows the CW4011 CPU registers. There are 32 general registers, each consisting of a single word (32 bits). The 32 general registers are treated symmetrically with two exceptions: *r0* is hardwired to a zero value and *r31* is defined as the link register for jump and link instructions.

Figure A.1 CW4011 CPU Registers



Register *r0* may be specified as a target register for any instruction when the result of the operation is discarded. The register maintains a value of zero under most conditions when used as a source register.

The two Multiply/Divide registers (HI, LO) store the doubleword, 64-bit result of multiply and divide operations.

A.2 Register Summary

Table A.1 lists the CW4011-specific registers, their location by either CP0 number or physical address, and the page number where each is described. All the registers listed in this section are separate from the general CPU registers listed in Section A.1, "CW4011 CPU Registers."

Table A.1 CP0 Exception Processing Registers

Register Name	CP0 Register Number	Physical Address	Reference Page
Context	4	—	4-5
Debug Control and Status (DCS)	7	—	4-7
Bad Virtual Address (BadVAddr)	8	—	4-9
Count	9	—	4-9
Compare	11	—	4-9
Status	12	—	4-10
Cause	13	—	4-18
Exception Program Counter (EPC)	14	—	4-20
Processor Revision Identifier (PRId)	15	—	4-20
Configuration and Cache Control (CCC)	16	—	4-22
Load Linked Address (LLAdr)	17	—	4-26
Breakpoint Program Counter (BPC)	18	—	4-27
(Sheet 1 of 2)			

Table A.1 CP0 Exception Processing Registers (Cont.)

Register Name	CP0 Register Number	Physical Address	Reference Page
Breakpoint Data Address (BDA)	19	—	4-27
Breakpoint PC Mask (BPCM)	20	—	4-27
Breakpoint Data Address Mask (BDAM)	21	—	4-28
Rotate	23	—	4-28
Circular Mask (CMask)	24	—	4-29
Error Exception Program Counter (Error EPC)	30	—	4-30
EntryHi	10	—	5-10
EntryLo	2	—	5-11
PageMask	5	—	5-12
Index	0	—	5-13
Random	1	—	5-13
Wired	6	—	5-14
Rx Status	—	0x10FF0000	9-10
Rx Setup	—	0x10FF0000	9-11
Rx Data	—	0x10FF0004	9-11
Tx Status	—	0x10FF0008	9-12
Tx Data	—	0x10FF000C	9-12
(Sheet 2 of 2)			

Appendix B
Cache Sizing and Design Concerns

This appendix describes the I-cache and D-cache sizing and design considerations in a CW4011 system and is split into the following sections:

B.1 CW4011 I-Cache Configurations

The CW4011 supports a number of two-way set associative and direct-mapped I-cache configurations, as shown in Table B.1.

Table B.1 CW4011 I-Cache Sizes

Two-Way Set Associative I-Cache (Kbytes)	Direct-Mapped I-Cache (Kbytes)	CCC Register IS[1:0] Settings
2	1	00
4	2	01
8	4	10
16	8	11

Different physical I-cache configurations require different sizes for tag RAM and data RAM, as well as different CW4011 core signal connections, as described in Section B.4, “I-Cache Set Associative RAM Hookup.”

However, usually a larger I-cache design can emulate any of the smaller available configurations through the IS[1:0] bits in the CCC register. For example, a 16-Kbyte I-cache can be configured to emulate all other possible configurations if the tag memory has enough bit tag width for small configurations. To test a 1-Kbyte configuration, the I-cache tag would need 23 bits. This I-cache size flexibility allows bench marking of various configurations in the simulation model and on the reference device chip. For more information on dynamic cache sizing, see Section 4.3.10, “Configuration and Cache Control (CCC) Register.”

The following sections describe the physical port interconnects between the CW4011 and the synchronous RAM models. This connection is performed in Verilog or VHDL at the shell level. In the CW4011 deliverables database, Verilog and VHDL are available as an example connection.

B.2 CW4011 I-Cache Interface

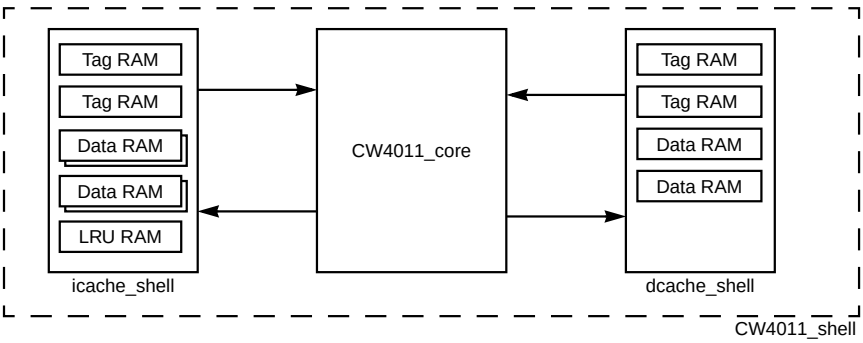
The CW4011 I-cache interface was designed to implement a two-way set associative I-cache of varying size. The I-cache uses a least recently used (LRU) algorithm to determine which set should be replaced when both cache lines are valid in the case of two-way set associative configuration. Logically, 256x1 memory is necessary. Since HDRAMs are not available in formats less than four bits wide, 64x4 memory is used as the LRU memory for 16-Kbyte systems.

For a list of the I-cache RAM interface signals, see Section 7.8, “Instruction Cache Interface.”

B.3 I-Cache Shell

In the CW4011 RTL shell model, the deliverables data, tag, and data memories are gathered in a shell module for the I-cache. The hierarchy is shown in Figure B.1. There are several glue logic gates in the icache_shell.

Figure B.1 CW4011 I-Cache Shell RTL



B.4 I-Cache Set Associative RAM Hookup

I-cache set associative implementations require:

- two tag RAMs
- four data RAMs
- one LRU RAM

Table B.2 lists the required RAM size.

Table B.2 Set Associative, I-Cache RAM Requirements

Set Associative I-Cache Size (Kbytes)	Tag RAM		Data RAM		LRU RAM[1]	
	Quantity	Size (Bits)	Quantity	Size (Bits)	Quantity	Size (Bits)
2	2	32x23	4	128x32	1	32x1
4	2	64x22	4	256x32	1	64x1
8	2	128x21	4	512x32	1	128x1
16	2	256x20	4	1024x32	1	256x1

1. Listed are the logic needs for the LRU RAM. SInce the minimum size for an HDRAM is 64x4, one 64x4 HDRAM can be used for any of the I-cache size configurations. The deliverables database has an HDL model named “fake256x1,” which renames the names of signals of a 64x4 RAM to those of a 256x1.

The RAMs used should be word write enabled, synchronous RAMs such as the m10p111hs for the CW4011 (LCBG10P).

The following sections describe the connections from the CW4011 core I-cache interface ports to the I-cache RAM macros ports, assuming m10p111hs RAMs are implemented. Unspecified I/O can be considered unconnected. Unconnected core inputs should be tied deasserted. Unconnected core outputs may be left open.

For bigger cache sizes, the CW4011 core needs fewer tag inputs. Unused tag inputs are ignored according to the programming of the CCC register.

If both or either cache set is used as a RAM where the address is fixed in the memory address space permanently, the tag memory is not necessary. The tag inputs must be tied either LOW or HIGH, according to the specific design address mapping.

B.4.1 2-Kbyte I-Cache Set Associative Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made. Connections for LRU memory “fake256x1” are not shown. Refer the HDL model of the fake256x1 for more information.

B.4.1.1 Tag RAM Set 0

DO[22:0] ↔ iC0TAGDop[22:0]
DI[22:0] ↔ iCTAGDip[22:0]
A[4:0] ↔ iTADDRp[4:0]
(open) ↔ iTADDRp[7:5]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.1.2 Tag RAM Set 1

DO[22:0] ↔ iC1TAGDop[22:0]
DI[22:0] ↔ iCTAGDip[22:0]
A[4:0] ↔ iTADDRp[4:0]
(open) ↔ iTADDRp[7:5]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe1p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.1.3 Data RAM Set 0 High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.1.4 Data RAM Set 0 Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.1.5 Data RAM Set 1 High Word

DO[31:0] ↔ iC1DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.1.6 Data RAM Set 1 Low Word

DO[31:0] ↔ iC1DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2 4-Kbyte I-Cache Set Associative Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made. Connections for LRU memory “fake256x1” are not shown.

B.4.2.1 Tag RAM Set 0

DO[21:0] ↔ iC0TAGDop[22:1]
1’b0 ↔ iC0TAGDop[1]
DI[21:0] ↔ iCTAGDip[22:1]
(open) ↔ iCTAGDip[1]
A[5:0] ↔ iTADDRp[5:0]
(open) ↔ iTADDRp[7:6]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2.2 Tag RAM Set 1

DO[21:0] ↔ iC1TAGDop[22:1]
1’b0 ↔ iC1TAGDop[1]
DI[21:0] ↔ iCTAGDip[22:1]
(open) ↔ iCTAGDip[1]
A[5:0] ↔ iTADDRp[5:0]
(open) ↔ iTADDRp[7:6]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe1p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2.3 Data RAM Set 0 High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp

WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2.4 Data RAM Set 0 Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2.5 Data RAM Set 1 High Word

DO[31:0] ↔ iC1DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.2.6 Data RAM Set 1 Low Word

DO[31:0] ↔ iC1DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3 8-Kbyte, Set Associative Cache, Hookup

The following connections between the CW4011 core and the synchronous RAM modules need to be made. Connections for LRU memory “fake256x1” are not shown.

B.4.3.1 Tag RAM Set 0

DO[20:0] ↔ iC0TAGDop[22:2]
2’b0↔ iC0TAGDop[2:1]
DI[20:0] ↔ iCTAGDip[22:2]
(open) ↔ iCTAGDip[2:1]
A[6:0] ↔ iTADDRp[6:0]
(open) ↔ iTADDRp[7]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3.2 Tag RAM Set 1

DO[20:0] ↔ iC1TAGDop[22:2]
2’b0↔ iC1TAGDop[2:1]
DI[20:0] ↔ iCTAGDip[22:2]
(open) ↔ iCTAGDip[2:1]
A[6:0] ↔ iTADDRp[6:0]
(open) ↔ iTADDRp[7]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe1p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3.3 Data RAM Set 0 High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp

WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3.4 Data RAM Set 0 Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3.5 Data RAM Set 1 High Word

DO[31:0] ↔ iC1DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.3.6 Data RAM Set 1 Low Word

DO[31:0] ↔ iC1DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4 16-Kbyte I-Cache Set Associative Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made. Connections for LRU memory “fake256x1” are not shown. Refer the HDL model of the fake256x1 for more information.

B.4.4.1 Tag RAM Set 0

DO[19:0] ↔ iC0TAGDop[22:3]
3’b0↔ iC0TAGDop[3:1]
DI[19:0] ↔ iCTAGDip[22:3]
(open) ↔ iCTAGDip[3:1]
A[7:0] ↔ iTADDRp[7:0]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4.2 Tag RAM Set 1

DO[19:0] ↔ iC1TAGDop[22:3]
3’b0↔ iC1TAGDop[3:1]
DI[19:0] ↔ iCTAGDip[22:3]
(open) ↔ iCTAGDip[3:1]
A[7:0] ↔ iTADDRp[7:0]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe1p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4.3 Data RAM Set 0 High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4.4 Data RAM Set 0 Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4.5 Data RAM Set 1 High Word

DO[31:0] ↔ iC1DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.4.4.6 Data RAM Set 1 Low Word

DO[31:0] ↔ iC1DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5 I-Cache Direct-Mapped RAM

Direct-mapped I-cache implementations require one tag RAMs and two data RAMs. Table B.3 lists the required RAM sizes.

Table B.3 Direct-Mapped, WriteBack, I-Cache RAM Requirements

Direct-Mapped I-Cache Size (Kbytes)	Tag RAM		Data RAM	
	Quantity	Size (Bits)	Quantity	Size (Bits)
1	1	32x23	2	128x32
2	1	64x22	2	256x32
4	1	128x21	2	512x32
8	1	256x20	2	1024x32

The RAMs used should be word write enabled, synchronous RAMs such as the m10p111hs for the CW4011 (LCBG10P).

In the case of direct-mapped cache configuration, inputs of one set are not used. All unused inputs are ignored internally, but they should be always be tied deasserted.

The following sections describe the connections from the CW4011 core I-cache interface ports to the I-cache RAM macros ports, assuming m10p111hs RAMs. Unspecified I/O can be considered unconnected. Of course, unconnected core inputs should be tied deasserted.

B.5.1 1-Kbyte I-Cache Direct-Mapped Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.5.1.1 Tag RAM

DO[22:0] ↔ iC0TAGDop[22:0]
DI[22:0] ↔ iCTAGDip[22:0]
A[4:0] ↔ iTADDRp[4:0]
(open) ↔ iTADDRp[7:5]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.1.2 Data RAM High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.1.3 Data RAM Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[6:0] ↔ iCADDRp[6:0]
(open) ↔ iCADDRp[9:7]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.2 2-Kbyte I-Cache Direct-Mapped Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.5.2.1 Tag RAM

DO[21:0] ↔ iC0TAGDop[22:1]
1'b0 ↔ iC0TAGDop[1]
DI[21:0] ↔ iCTAGDip[22:1]
(open) ↔ iCTAGDip[1]
A[5:0] ↔ iTADDRp[5:0]
(open) ↔ iTADDRp[7:6]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.2.2 Data RAM High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.2.3 Data RAM Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[7:0] ↔ iCADDRp[7:0]
(open) ↔ iCADDRp[9:8]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.3 4-Kbyte I-Cache Direct-Mapped Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.5.3.1 Tag RAM

DO[20:0] ↔ iC0TAGDop[22:2]
2'b0↔ iC0TAGDop[2:1]
DI[20:0] ↔ iCTAGDip[22:2]
(open) ↔ iCTAGDip[2:1]
A[6:0] ↔ iTADDRp[6:0]
(open) ↔ iTADDRp[7]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.3.2 Data RAM High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.3.3 Data RAM Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.4 8-Kbyte I-Cache Direct-Mapped Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.5.4.1 Tag RAM

DO[19:0] ↔ iC0TAGDop[22:3]
3'b0↔ iC0TAGDop[3:1]
DI[19:0] ↔ iCTAGDip[22:3]
(open) ↔ iCTAGDip[3:1]
A[7:0] ↔ iTADDRp[7:0]
CLK ↔ Shell Clock
OE ↔ iCTAGRDp
WE ↔ ITWe0p (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.4.2 Data RAM High Word

DO[31:0] ↔ iC0DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.4.3 Data RAM Low Word

DO[31:0] ↔ iC0DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[9:0] ↔ iCADDRp[9:0]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe0lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.5 Instruction RAM

I-cache Set 1 can be used as a scratchpad RAM. The I-cache scratchpad RAM is enabled 7by setting the ISR1 bit in the CCC register to one.

If the address space is fixed permanently, the tag memory for I-cache Set 1 is not necessary. The tag inputs must be tied LOW or HIGH, according to a design's address mapping.

If the address space should be programmable, the tag memory must be initialized before I-cache Set 1 is used as an instruction RAM.

For both cases, instruction codes for the instruction RAM must be written to the instruction data RAM of Set 1 by a cache maintenance function, which is enabled by Isolate Cache (IsC) and Tag bits of the CCC register. For more information, see Section 4.3.10, "Configuration and Cache Control (CCC) Register."

The following is an example of a 4-Kbyte instruction RAM configuration without a tag.

B.5.5.1 Tag RAM Set 1

DO[31:12] ↔ iC0TAGDop[21:2]
1'b1 ↔ iC0TAGDop[22]
2'b00 ↔ iC0TAGDop[1:0]
(open)↔ iCTAGDip[22:0]
(open) ↔ iTADDRp[7:0]
(open) ↔ iCTAGRDp
(open) ↔ ITWe0p (Note: Generated by icache_shell glue logic)
CLK ↔ Shell Clock
Enable ↔ HIGH

B.5.5.2 Data RAM High Word

DO[31:0] ↔ iC1DATADohp[31:0]
DI[31:0] ↔ iCDATADip[63:32]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp

WE ↔ ICWe1hp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.5.5.3 Data RAM Low Word

DO[31:0] ↔ iC1DATADolp[31:0]
DI[31:0] ↔ iCDATADip[31:0]
A[8:0] ↔ iCADDRp[8:0]
(open) ↔ iCADDRp[9]
CLK ↔ Shell Clock
OE ↔ iCDATARDp
WE ↔ ICWe1lp (Note: Generated by icache_shell glue logic)
Enable ↔ HIGH

B.6 CW4011 D-Cache Configurations

The CW4011 supports a number of two-way set associative and direct-mapped D-cache configurations, as shown in Table B.4.

Table B.4 CW4011 D-Cache Sizes

Two-way Set Associative D-Cache (Kbytes)	Direct-Mapped D-Cache (Kbytes)	CCC Register DS[1:0] Settings
2	1	00
4	2	01
8	4	10
16	8	11

Different physical D-cache configurations require different sizes for tag RAM and data RAM, as well as different CW4011 core signal connections, as described in Section B.9, “D-Cache Set Associative RAM Hookup.”

However, usually a larger D-cache design can emulate any of the smaller available configurations through the DS[1:0] bits in the CCC register. For example, a 16-Kbyte D-cache can be configured to emulate all other possible configurations if the tag memory has enough bit tag width for

small configurations. To test a 1-Kbyte configuration, the D-cache tag would need 24 bits. This D-cache size flexibility allows bench marking of various configurations in the simulation model and on the reference device chip. For more information on dynamic cache sizing, see Section 4.3.10, “Configuration and Cache Control (CCC) Register.”

The following sections describe the physical port interconnect between the CW4011 and the synchronous RAM models. This connection is in Verilog or VHDL at the shell level. In the CW4011 deliverables database, Verilog and VHDL models are available as an example.

B.7 CW4011 D-Cache Interface

The CW4011 D-cache interface was designed to implement a two-way set associative D-cache of varying size. In the CW4011, the data associativities (Set 0 and Set 1) are interleaved across two data RAMs (Bank A and Bank B). This allows 64 bit reads and writes to the SCbus on cache refills, while only requiring two banks of 32-bit wide RAMs. Table B.5 shows this information interleaving.

Table B.5 D-Cache Data Interleaving

Bank A	Bank B
Line 0, Set 0, Word 0	Line 0, Set 1 Word 0
Line 0, Set 1, Word 1	Line 0, Set 0 Word 1
Line 0, Set 0, Word 2	Line 0, Set 1 Word 2
Line 0, Set 1, Word 3	Line 0, Set 0 Word 3
Line 0, Set 0, Word 4	Line 0, Set 1 Word 4
Line 0, Set 1, Word 5	Line 0, Set 0 Word 5
Line 0, Set 0, Word 6	Line 0, Set 1 Word 6
Line 0, Set 1, Word 7	Line 0, Set 0 Word 7
Line 1, Set 0, Word 0	Line 1, Set 1 Word 0
Line 1, Set 1, Word 1	Line 1, Set 0 Word 1
Line 1, Set 0, Word 2	Line 1, Set 1 Word 2
Line 1, Set 1, Word 3	Line 1, Set 0 Word 3
Line 1, Set 0, Word 4	Line 1, Set 1 Word 4
(Sheet 1 of 2)	

Table B.5 D-Cache Data Interleaving (Cont.)

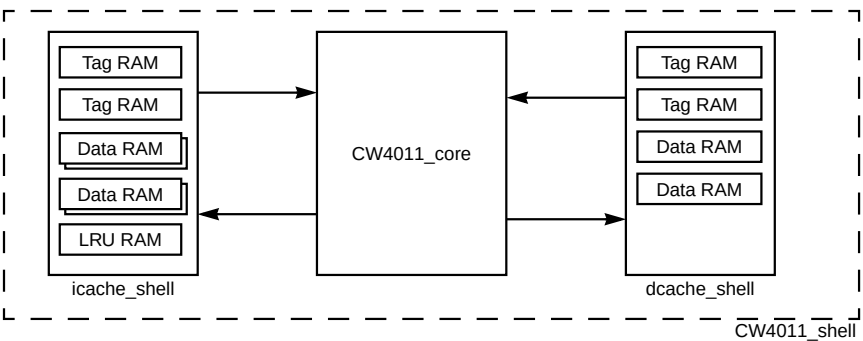
Bank A	Bank B
Line 1, Set 1, Word 5	Line 1, Set 0 Word 5
Line 1, Set 0, Word 6	Line 1, Set 1 Word 6
Line 1, Set 1, Word 7	Line 1, Set 0 Word 7
. . .	. . .
Line n, Set 0, Word 4	Line n, Set 1 Word 4
Line n, Set 1, Word 5	Line n, Set 0 Word 5
Line n, Set 0, Word 6	Line n, Set 1 Word 6
Line n, Set 1, Word 7	Line n, Set 0 Word 7
(Sheet 2 of 2)	

For more detailed information about the CW4011 D-cache interface signals, see Section 7.7, “Data Cache Interface.”

B.8 D-Cache Shell

In the CW4011 RTL shell model, the deliverables data, tag, and data memories are gathered in a shell module for D-cache. Figure B.2 shows the CW4011_shell hierarchy.

Figure B.2 CW4011 D-Cache Shell RTL



B.9 D-Cache Set Associative RAM Hookup

Set associative implementations require two tag RAMs and two data RAMs. Table B.6 and Table B.7 list the required RAM sizes.

Table B.6 Set Associative, WriteBack, D-Cache RAM Requirements

Set Associative I-Cache Size (Kbytes)	Tag RAM		Data RAM	
	Quantity	Size (Bits)	Quantity	Size (Bits)
2	2	32x24	2	256x32
4	2	64x23	2	512x32
8	2	128x22	2	1024x32
16	2	256x21	2	2048x32

Table B.7 Set Associative, WriteThrough, D-Cache RAM Requirements

Set Associative I-Cache Size (Kbytes)	Tag RAM		Data RAM	
	Quantity	Size (Bits)	Quantity	Size (Bits)
2	2	32x23	2	256x32
4	2	64x22	2	512x32
8	2	128x21	2	1024x32
16	2	256x20	2	2048x32

The RAMs used should be word write enabled, synchronous RAMs such the m10p111hs for the CW4011 (LCBG10P).

The following sections describe the connections from the CW4011 core D-cache interface ports to the D-cache RAM macros ports, assuming m10p111hs RAMs. Unspecified I/O can be considered unconnected. Of course, unconnected core inputs should be tied deasserted. Unconnected core outputs may be left open.

For bigger cache size, the CW4011 core needs fewer tag inputs. Unused tag inputs are ignored, according to the programming of the CCC register.

If both or either cache set is used as a RAM of which address is fixed in the memory address space permanently, the tag memory is not

necessary. The tag inputs must be tied either LOW or HIGH, according to the address mapping.

Refer the HDL model of the cache shell for details about the connections to memory macros.

B.9.1 2-Kbyte D-Cache Set Associative, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.1.1 Tag RAM Set 0

DO[23:0] ↔ DC0TAGDop[23:0]
DI[23:0] ↔ DC0TAGDip[23:0]
A[4:0] ↔ DCTAGADDRp[9:5]]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock
OE[23:0] ↔ HIGH
WE[23:0] ↔ {23{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.9.1.2 Tag RAM Set 1

DO[23:0] ↔ DC1TAGDop[23:0]
DI[23:0] ↔ DC1TAGDip[23:0]
A[4:0] ↔ DCTAGADDRp[9:5]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock
OE[23:0] ↔ HIGH
WE[23:0] ↔ {23{DTWeBp[1]},DTWeBp[0]}
Enable ↔ HIGH

B.9.1.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[7:0] ↔ {DCDATAADDRp[9:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0]↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.1.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[7:0] ↔ {DCDATAADDRp[9:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.2 4-Kbyte D-Cache Set Associative, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.2.1 Tag RAM Set 0

DO[22:0] ↔ {DC0TAGDop[23:3],DC0TAGDop[1:0]}
1'b0 ↔ DC0TAGDop[2]
DI[22:0] ↔ {DC0TAGDip[23:3],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[22:0] ↔ HIGH
WE[22:0] ↔ {22{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.9.2.2 Tag RAM Set 1

DO[22:0] ↔ {DC1TAGDop[23:3],DC1TAGDop[1:0]}
1'b0 ↔ DC1TAGDop[2]
DI[22:0] ↔ {DC1TAGDip[23:3],DC1TAGDip[1:0]}
(open) ↔ DC0TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[22:0] ↔ HIGH
WE[22:0] ↔ {22{DTWeBp[1]},DTWeBp[0]}
Enable ↔ HIGH

B.9.2.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[8:0] ↔ {DCDATAADDRp[10:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.2.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[8:0] ↔ {DCDATAADDRp[10:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.3 8-Kbyte D-Cache Set Associative, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.3.1 Tag RAM Set 0

DO[21:0] ↔ {DC0TAGDop[23:4],DC0TAGDop[1:0]}
2'b0 ↔ DC0TAGDop[3:2]
DI[21:0] ↔ {DC0TAGDip[23:4],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ {21{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.9.3.2 Tag RAM Set 1

DO[21:0] ↔ {DC1TAGDop[23:4],DC1TAGDop[1:0]}
2'b0 ↔ DC1TAGDop[3:2]
DI[21:0] ↔ {DC1TAGDip[23:4],DC1TAGDip[1:0]}
(open) ↔ DC1TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ {21{DTWeBp[1]},DTWeBp[0]}
Enable ↔ HIGH

B.9.3.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[9:0] ↔ {DCDATAADDRp[11:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔

{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.3.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[9:0] ↔ {DCDATAADDRp[11:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.4 16-Kbyte D-Cache Set Associative, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.4.1 Tag RAM Set 0

DO[20:0] ↔ {DC0TAGDop[23:5],DC0TAGDop[1:0]}
3'b0 ↔ DC0TAGDop[4:2]
DI[20:0] ↔ {DC0TAGDip[23:5],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH
WE[20:0] ↔ {20{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.9.4.2 Tag RAM Set 1

DO[20:0] ↔ {DC1TAGDop[23:5],DC1TAGDop[1:0]}
3'b0 ↔ DC1TAGDop[4:2]
DI[20:0] ↔ {DC1TAGDip[23:5],DC1TAGDip[1:0]}
(open) ↔ DC1TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH

WE[20:0] ↔ {20{DTWeBp[1]},DTWeBp[0]}
Enable ↔ HIGH

B.9.4.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[10:0] ↔ {DCDATAADDRp[12:3],DCADATAADDRp}
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.4.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[10:0] ↔ {DCDATAADDRp[12:3],DCBDATAADDRp}
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.5 2-Kbyte D-Cache Set Associative, WriteThrough Connections

For write through cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.5.1 Tag RAM Set 0

DO[22:0] ↔ {DC0TAGDop[23:1]}
1'b0 ↔ DC0TAGDop[0]
DI[22:0] ↔ DC0TAGDip[23:1]
(open) ↔ DC0TAGDip[0]
A[4:0] ↔ DCTAGADDRp[9:5]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock

OE[22:0] ↔ HIGH
WE[22:0] ↔ 23{DTWeAp[1]}
Enable ↔ HIGH

B.9.5.2 Tag RAM Set 1

DO[22:0] ↔ {DC1TAGDop[23:1]}
1'b0 ↔ DC1TAGDop[0]
DI[22:0] ↔ DC1TAGDip[23:1]
(open) ↔ DC1TAGDip[0]
A[4:0] ↔ DCTAGADDRp[9:5]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock
OE[22:0] ↔ HIGH
WE[22:0] ↔ 23{DTWeBp[1]}
Enable ↔ HIGH

B.9.5.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[7:0] ↔ {DCDATAADDRp[9:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.5.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[7:0] ↔ {DCDATAADDRp[9:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.6 4-Kbyte D-Cache Set Associative, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.6.1 Tag RAM Set 0

DO[21:0] ↔ {DC0TAGDop[23:3],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
1'b0 ↔ DC0TAGDop[2]
DI[21:0] ↔ {DC0TAGDip[23:3],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ 22{DTWeAp[1]}
Enable ↔ HIGH

B.9.6.2 Tag RAM Set 1

DO[21:0] ↔ {DC1TAGDop[23:3],DC1TAGDop[1]}
1'b0 ↔ DC1TAGDop[0]
1'b0 ↔ DC1TAGDop[2]
DI[21:0] ↔ {DC1TAGDip[23:3],DC1TAGDip[1]}
(open) ↔ DC1TAGDip[0]
(open) ↔ DC1TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ 22{DTWeBp[1]}
Enable ↔ HIGH

B.9.6.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[8:0] ↔ {DCDATAADDRp[10:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.6.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[8:0] ↔ {DCDATAADDRp[10:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.7 8-Kbyte D-Cache Set Associative, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.7.1 Tag RAM Set 0

DO[20:0] ↔ {DC0TAGDop[23:4],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
2'b0 ↔ DC0TAGDop[3:2]
DI[20:0] ↔ {DC0TAGDip[23:4],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH
WE[20:0] ↔ 21{DTWeAp[1]}
Enable ↔ HIGH

B.9.7.2 Tag RAM Set 1

DO[20:0] ↔ {DC1TAGDop[23:4],DC1TAGDop[1]}
1'b0 ↔ DC1TAGDop[0]
2'b0 ↔ DC1TAGDop[3:2]
DI[20:0] ↔ {DC1TAGDip[23:4],DC1TAGDip[1]}
(open) ↔ DC1TAGDip[0]
(open) ↔ DC1TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH
WE[20:0] ↔ 21{DTWeBp[1]}
Enable ↔ HIGH

B.9.7.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[9:0] ↔ {DCDATAADDRp[11:3],DCADATAADDRp}
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.7.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[9:0] ↔ {DCDATAADDRp[11:3],DCBDATAADDRp}
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.9.8 16-Kbyte D-Cache Set Associative, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.9.8.1 Tag RAM Set 0

DO[19:0] ↔ {DC0TAGDop[23:5],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
3'b0 ↔ DC0TAGDop[4:2]
DI[19:0] ↔ {DC0TAGDip[23:5],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[19:0] ↔ HIGH
WE[19:0] ↔ 20{DTWeAp[1]}
Enable ↔ HIGH

B.9.8.2 Tag RAM Set 1

DO[19:0] ↔ {DC1TAGDop[23:5],DC1TAGDop[1]}
1'b0 ↔ DC1TAGDop[0]
3'b0 ↔ DC1TAGDop[4:2]
DI[19:0] ↔ {DC1TAGDip[23:5],DC1TAGDip[1]}
(open) ↔ DC1TAGDip[0]
(open) ↔ DC1TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[19:0] ↔ HIGH
WE[19:0] ↔ 20{DTWeBp[1]}
Enable ↔ HIGH

B.9.8.3 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[10:0] ↔ {DCDATAADDRp[12:3],DCADATAADDRp}
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.9.8.4 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[10:0] ↔ {DCDATAADDRp[12:3],DCBDATAADDRp}
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10 D-Cache Direct-Mapped RAM Hookup

Direct-mapped D-cache implementations require one tag RAM and two data RAMs. Table B.8 and Table B.9 list the required RAM sizes.

Table B.8 Direct-Mapped, WriteBack, D-Cache RAM Requirements

Direct-Mapped I-Cache Size (Kbytes)	Tag RAM		Data RAM	
	Quantity	Size (Bits)	Quantity	Size (Bits)
1	1	32x24	2	128x32
2	1	64x23	2	256x32
4	1	128x22	2	512x32
8	1	256x21	2	1024x32

Table B.9 Direct-Mapped, WriteThrough, D-Cache RAM Requirements

Direct-Mapped I-Cache Size (Kbytes)	Tag RAM		Data RAM	
	Quantity	Size (Bits)	Quantity	Size (Bits)
1	1	32x23	2	128x32
2	1	64x22	2	256x32
4	1	128x21	2	512x32
8	1	256x20	2	1024x32

The RAMs used should be word write enabled, synchronous RAMs such as the m10p111hs for the CW4011 (LCBG10P).

In the case of direct-mapped cache configuration, inputs of one set are not used. All unused inputs are ignored internally, but they should be tied deasserted.

The following sections describe connections from the CW4011 core D-cache interface ports to the D-cache RAM macros ports assuming m10p111hs RAMs. Unspecified I/O can be considered unconnected. Of course, unconnected core inputs should be tied deasserted.

B.10.1 1-Kbyte D-Cache Direct-Mapped, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.1.1 Tag RAM Set 0

DO[23:0] ↔ DC0TAGDop[23:0]
DI[23:0] ↔ DC0TAGDip[23:0]
A[4:0] ↔ DCTAGADDRp[9:5]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock
OE[23:0] ↔ HIGH
WE[23:0] ↔ {23{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.10.1.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[6:0] ↔ DCDATAADDRp[9:3]
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.1.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[6:0] ↔ DCDATAADDRp[9:3]
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.2 2-Kbyte D-Cache Direct-Mapped, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.2.1 Tag RAM Set 0

DO[22:0] ↔ {DC0TAGDop[23:3],DC0TAGDop[1:0]}
1'b0 ↔ DC0TAGDop[2]
DI[22:0] ↔ {DC0TAGDip[23:3],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[22:0] ↔ HIGH
WE[22:0] ↔ {22{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.10.2.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[7:0] ↔ DCDATAADDRp[10:3]
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.2.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[7:0] ↔ DCDATAADDRp[10:3]
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.3 4-Kbyte D-Cache Direct-Mapped, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.3.1 Tag RAM Set 0

DO[21:0] ↔ {DC0TAGDop[23:4],DC0TAGDop[1:0]}
2'b0 ↔ DC0TAGDop[3:2]
DI[21:0] ↔ {DC0TAGDip[23:4],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ {21{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.10.3.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[8:0] ↔ DCDATAADDRp[11:3]
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.3.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[8:0] ↔ {DCDATAADDRp[11:3]
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.4 8-Kbyte D-Cache Direct-Mapped, WriteBack Connections

The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.4.1 Tag RAM Set 0

DO[20:0] ↔ {DC0TAGDop[23:5],DC0TAGDop[1:0]}
3'b0 ↔ DC0TAGDop[4:2]
DI[20:0] ↔ {DC0TAGDip[23:5],DC0TAGDip[1:0]}
(open) ↔ DC0TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH
WE[20:0] ↔ {20{DTWeAp[1]},DTWeAp[0]}
Enable ↔ HIGH

B.10.4.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.4.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.5 1-Kbyte D-Cache Direct-Mapped, WriteThrough Connections

For write through cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.5.1 Tag RAM Set 0

DO[22:0] ↔ {DC0TAGDop[23:1]}
1'b0 ↔ DC0TAGDop[0]
DI[22:0] ↔ DC0TAGDip[23:1]
(open) ↔ DC0TAGDip[0]
A[4:0] ↔ DCTAGADDRp[9:5]
(open) ↔ DCTAGADDRp[12:10]
CLK ↔ Shell Clock
OE[22:0] ↔ HIGH
WE[22:0] ↔ 23{DTWeAp[1]}
Enable ↔ HIGH

B.10.5.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[6:0] ↔ DCDATAADDRp[9:3]
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.5.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[6:0] ↔ DCDATAADDRp[9:3]
(open) ↔ DCDATAADDRp[12:10]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.6 2-Kbyte D-Cache Direct-Mapped, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.6.1 Tag RAM Set 0

DO[21:0] ↔ {DC0TAGDop[23:3],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
1'b0 ↔ DC0TAGDop[2]
DI[21:0] ↔ {DC0TAGDip[23:3],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[2]
A[5:0] ↔ DCTAGADDRp[10:5]
(open) ↔ DCTAGADDRp[12:11]
CLK ↔ Shell Clock
OE[21:0] ↔ HIGH
WE[21:0] ↔ 22{DTWeAp[1]}
Enable ↔ HIGH

B.10.6.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[7:0] ↔ DCDATAADDRp[10:3]
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.6.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[7:0] ↔ DCDATAADDRp[10:3]
(open) ↔ DCDATAADDRp[12:11]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.7 4-Kbyte D-Cache Direct-Mapped, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.7.1 Tag RAM Set 0

DO[20:0] ↔ {DC0TAGDop[23:4],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
2'b0 ↔ DC0TAGDop[3:2]
DI[20:0] ↔ {DC0TAGDip[23:4],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[3:2]
A[6:0] ↔ DCTAGADDRp[11:5]
(open) ↔ DCTAGADDRp[12]
CLK ↔ Shell Clock
OE[20:0] ↔ HIGH
WE[20:0] ↔ 21{DTWeAp[1]}
Enable ↔ HIGH

B.10.7.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[8:0] ↔ DCDATAADDRp[11:3]
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.7.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[8:0] ↔ DCDATAADDRp[11:3]
(open) ↔ DCDATAADDRp[12]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
    {8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.8 8-Kbyte D-Cache Direct-Mapped, WriteThrough Connections

For WriteThrough cache, the WB bit in the tag is unused. The DC0TAGDop[0] and DC1TAGDop[0] inputs of the core should be tied LOW. The following connections between the CW4011 core and the synchronous RAM modules need to be made.

B.10.8.1 Tag RAM Set 0

DO[19:0] ↔ {DC0TAGDop[23:5],DC0TAGDop[1]}
1'b0 ↔ DC0TAGDop[0]
3'b0 ↔ DC0TAGDop[4:2]
DI[19:0] ↔ {DC0TAGDip[23:5],DC0TAGDip[1]}
(open) ↔ DC0TAGDip[0]
(open) ↔ DC0TAGDip[4:2]
A[7:0] ↔ DCTAGADDRp[12:5]
CLK ↔ Shell Clock
OE[19:0] ↔ HIGH
WE[19:0] ↔ 20{DTWeAp[1]}
Enable ↔ HIGH

B.10.8.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.8.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔

{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

B.10.9 Data Scratchpad RAM

Both or either D-cache set can be used as data scratchpad RAM, where the address is fixed in the memory address space. It is enabled by setting either the SR0 bit to one for Set 0, or the SR1 bit to one for Set 1, Both the SR0 and SR1 bits are in the CCC register.

If the address space is fixed permanently, the tag memory is not necessary. The tag inputs must be tied either LOW or HIGH according to the address mapping.

If the address space should be programmable, the tag memory must be initialized to valid with appropriate address before used as a scratchpad RAM by a cache maintenance function, which is enabled by Isolate Cache (IsC) and Tag bits of the CCC register. For more information, see Section 4.3.10, “Configuration and Cache Control (CCC) Register.”

The following shows an example which has 8 Kbytes of scratchpad RAM only.

B.10.9.1 Tag RAM Set 0

DO[31:13] ↔ {DC0TAGDop[23:5],DC0TAGDop[1]}
5’b00010 ↔ DC0TAGDop[4:0]
(open) ↔ DC0TAGDip[23:0]
(open) ↔ DTWeAp[1]
Enable ↔ HIGH

B.10.9.2 Data RAM Bank A

DO[31:0] ↔ DCADATADop[31:0]
DI[31:0] ↔ DCADATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeAp[3]},8{DCWeAp[2]},8{DCWeAp[1]},8{DCWeAp[0]}}
Enable ↔ HIGH

B.10.9.3 Data RAM Bank B

DO[31:0] ↔ DCBDATADop[31:0]
DI[31:0] ↔ DCBDATADip[31:0]
A[9:0] ↔ DCDATAADDRp[12:3]
CLK ↔ Shell Clock
OE[31:0] ↔ HIGH
WE[31:0] ↔
{8{DCWeBp[3]},8{DCWeBp[2]},8{DCWeBp[1]},8{DCWeBp[0]}}
Enable ↔ HIGH

Appendix C
Programmer's Notes

This appendix contains information that will be useful if you are writing software for the CW4011 core. The information is arranged in functional groups: instruction related, CP0 or TLB related, and cache related.

C.1 Instruction-Related Notes

- The instruction prior to an ERET must not generate an exception. You can use a NOP (no operation) to make sure that this restriction is met.
- The WAITI instruction must be followed by at least one NOP.
- Trap instructions must not be placed in branch delay slots.

C.2 CP0 or TLB–Related Notes

- When the CW4011 is operating in R3000 exception compatibility mode, the RFE (Restore From Exception) instruction clears the LL (load linked) bit. This is consistent with R4000 mode ERET operation.
- If a TLB is not present or enabled in the system, CP0 will reflect a Coprocessor Unusable exception if an attempt is made to execute any of the TLB maintenance instructions: TLBP, TLBR, TLBWI, TLBWR.
- TLB instructions (TLBP, TLBR, TLBWI, TLBWR) cannot be preceded or followed by a data access instruction (load or store) that requires target address translation, that is, *kseg*, *kseg2*.
- The instruction prior to a TLBWI or TLBWR instruction must not generate an exception. You can use a NOP to make sure that this restriction is met.

- Three instructions are required between a MTC0 instruction that targets any of the TLB support registers (that is, EntryHi, EntryLo, PageMask, and Index) and a subsequent TLBWI or TLBWR instruction. This ensures that the results of the prior MTC0 instruction will be seen by the TLB write operation
- Five instructions are required between a MTC0 Status register operation that updates the coprocessor usability field (Status[31:28]) and a subsequent coprocessor instruction that expects to see the updated value.
- Seven instructions are required between a MTC0 EPC register operation and a subsequent ERET instruction that expects to see the updated value.

C.3 Cache-Related Notes

- When the CW4011 is operating in Isolate Cache mode, load and store operations to the cache are not allowed in the delay slot of Branch Likely instructions.

C.4 CW33300 Compatible Debug Extension Notes

- The existing CW33300 has some extensions to the CP0 that provide enhanced debugging and exception handler support. The CW4011 core remains compatible with these enhancements. Refer to the *CW33300 Enhanced Self-Embedding Processor Core User's Manual* for further information.

Glossary

Big-endian This is a method of data formatting in which each field is addressed by referring to its most-significant byte. This means that if you are accessing a four-byte, singleword, the most-significant byte is byte 03, and the most-significant bit is bit 31. See also Little-endian.

Bus sizing Refers to the ability of the processor to support and interface with data buses of different sizes.

Bus snooping This is the method used by the cache controller to monitor memory accesses performed by other bus masters.

Direct-map caching In a direct-mapped cache, each memory location is mapped to one position in the cache. Direct mapping is useful if you are storing small loops and sequential operations. You can use this type of caching for both the D-cache and the I-cache.

Encrypted Encrypted files are source code files that have been processed in a language such as HDL or Verilog so that they are only machine readable. This process enables you to have access to the behavior of the files but not to the intellectual property associated with them.

Fixup cycle This is a clock cycle during which the Load Miss data is funneled back to the instruction that requires it.

Little-endian This is a method of data formatting in which each field is addressed by referring to its least-significant byte. This means that if you are accessing a four-byte, singleword, the most-significant byte is byte 00, and the most-significant bit is bit 00. The CW4011 supports both little-endian and big-endian formats. See also Big-endian**.**

Placement algorithms Information is placed in a cache using placement algorithms. These algorithms define the positions in the cache where the information from a particular memory location may be stored. The CW4011 uses two types of algorithm, direct mapping and two-way set associative mapping.

Slip condition A slip condition occurs when the pipeline stalls after the EX stage. In this situation, the previous instruction is executed and clears the pipeline. However, the earlier stages of the pipeline are stalled.

Two-way set associative caching In a two-way set associative cache, each memory location is stored in one of two possible positions. Two-way set associative mapping is well-suited for data references, which tend to be more scattered than instructions. You can use this type of caching for both the D-cache and the I-cache.

Unencrypted Files that are unencrypted have not been subjected to the processing described in the entry Encrypted. These files are human readable and can be written using a text editor.

Verilog model Verilog is an open standard language. A Verilog model represents a design in the language. It provides no indication of the level of abstraction.

Customer Feedback

We would appreciate your feedback on this document. Please copy the following page, add your comments, and fax it to us at the address on the following page.

If appropriate, please also fax copies of any marked-up pages from this document.

Important: Please include your name, phone number, fax number, and company address so that we may contact you directly for clarification or additional information.

Thank you for your help in improving the quality of our documents.

Reader's Comments

Fax your comments to:

LSI Logic Corporation
Technical Publications
M/S E-198
Fax: 408.433.4333

Please tell us how you rate this document: *MiniRISC CW4011 Superscalar Microprocessor Core Technical Manual.* Place a check mark in the appropriate blank for each category.

	Excellent	**Good**	**Average**	**Fair**	**Poor**
Completeness of information	____	____	____	____	____
Clarity of information	____	____	____	____	____
Ease of finding information	____	____	____	____	____
Technical content	____	____	____	____	____
Usefulness of examples and illustrations	____	____	____	____	____
Overall manual	____	____	____	____	____

What could we do to improve this document?

_____

_____

_____

If you found errors in this document, please specify the error and page number. If appropriate, please fax a marked-up copy of the page(s).

_____

_____

_____

Please complete the information below so that we may contact you directly for clarification or additional information.

Name _____ Date _____

Telephone _____ Fax _____

Title _____

Department _____ Mail Stop _____

Company Name _____

Street _____

City, State, Zip _____

U.S. Distributors by State

H. H. Hamilton Hallmark
W. E. Wyle Electronics

Alabama
Huntsville
H. H. Tel: 205.837.8700
W. E. Tel: 800.964.9953

Alaska
H. H. Tel: 800.332.8638

Arizona
Phoenix
H. H. Tel: 602.736.7000
W. E. Tel: 800.528.4040
Tucson
H. H. Tel: 520.742.0515

Arkansas
H. H. Tel: 800.327.9989

California
Irvine
H. H. Tel: 714.789.4100
W. E. Tel: 800.626.9953
Los Angeles
H. H. Tel: 818.594.0404
W. E. Tel: 800.288.9953
Sacramento
H. H. Tel: 916.632.4500
W. E. Tel: 800.627.9953
San Diego
H. H. Tel: 619.571.7540
W. E. Tel: 800.829.9953
San Jose
H. H. Tel: 408.435.3500
Santa Clara
W. E. Tel: 800.866.9953
Woodland Hills
H. H. Tel: 818.594.0404

Colorado
Denver
H. H. Tel: 303.790.1662
W. E. Tel: 808.933.9953

Connecticut
Cheshire
H. H. Tel: 203.271.5700
Wallingford
W. E. Tel: 800.605.9953

Delaware
North/South
H. H. Tel: 800.526.4812
Tel: 800.638.5988

Florida
Fort Lauderdale
H. H. Tel: 305.484.5482
W. E. Tel: 800.568.9953
Orlando
H. H. Tel: 407.657.3300
W. E. Tel: 407.740.7450
N. Florida
W. E. Tel: 800.395.9953
St. Petersburg
H. H. Tel: 813.507.5000

Georgia
Atlanta
H. H. Tel: 770.623.4400
W. E. Tel: 800.876.9953

Hawaii
H. H. Tel: 800.851.2282

Idaho
H. H. Tel: 801.266.2022

Illinois
North/South
H. H. Tel: 847.797.7300
Tel: 314.291.5350
Chicago
W. E. Tel: 800.853.9953

Indiana
Indianapolis
H. H. Tel: 317.575.3500
W. E. Tel: 317.581.6152

Iowa
Cedar Rapids
H. H. Tel: 319.393.0033

Kansas
Kansas City
H. H. Tel: 913.663.7900

Kentucky
Central/Northern/ Western
H. H. Tel: 800.984.9503
Tel: 800.767.0329
Tel: 800.829.0146

Louisiana
North/South
H. H. Tel: 800.231.0253
Tel: 800.231.5575

Maine
H. H. Tel: 800.272.9255

Maryland
Baltimore
H. H. Tel: 410.720.3400
W. E. Tel: 800.863.9953

Massachusetts
Boston
H. H. Tel: 508.532.9808
W. E. Tel: 800.444.9953
Marlborough
W. E. Tel: 508.480.9900

Michigan
Detroit
H. H. Tel: 313.416.5800
Grandville
H. H. Tel: 616.531.0345

Minnesota
Minneapolis
H. H. Tel: 612.881.2600
W. E. Tel: 800.860.9953

Mississippi
H. H. Tel: 800.633.2918

Missouri
St. Louis
H. H. Tel: 314.291.5350

Montana
Bozeman
H. H. Tel: 800.526.1741

Nebraska
H. H. Tel: 800.332.4375

Nevada
Las Vegas
H. H. Tel: 800.528.8471

New Hampshire
H. H. Tel: 800.272.9255

New Jersey
North/South
H. H. Tel: 201.515.1641
Tel: 609.222.6400
Pine Brook
W. E. Tel: 800.862.9953

New Mexico
Albuquerque
H. H. Tel: 505.293.5119

New York
Long Island
H. H. Tel: 516.434.7400
W. E. Tel: 800.861.9953
Rochester
H. H. Tel: 716.475.9130
W. E. Tel: 800.319.9953
Syracuse
H. H. Tel: 315.453.4000

North Carolina
Raleigh
H. H. Tel: 919.872.0712
W. E. Tel: 800.560.9953

North Dakota
H. H. Tel: 800.829.0116

Ohio
Cleveland
H. H. Tel: 216.498.1100
W. E. Tel: 800.763.9953
Dayton
H. H. Tel: 614.888.3313
W. E. Tel: 800.763.9953

Oklahoma
Tulsa
H. H. Tel: 918.459.6000

Oregon
Portland
H. H. Tel: 503.526.6200
W. E. Tel: 800.879.9953

Pennsylvania
Pittsburgh
H. H. Tel: 412.281.4150
Philadelphia
H. H. Tel: 800.526.4812
W. E. Tel: 800.871.9953

Rhode Island
H. H. 800.272.9255

South Carolina
H. H. Tel: 919.872.0712

South Dakota
H. H. Tel: 800.829.0116

Tennessee
East/West
H. H. Tel: 800.241.8182
Tel: 800.633.2918

Texas
Austin
H. H. Tel: 512.219.3700
W. E. Tel: 800.365.9953
Dallas
H. H. Tel: 214.553.4300
W. E. Tel: 800.955.9953
El Paso
H. H. Tel: 800.526.9238
Houston
H. H. Tel: 713.781.6100
W. E. Tel: 800.888.9953
Rio Grande Valley
H. H. Tel: 210.412.2047

Utah
Draper
W. E. Tel: 800.414.4144
Salt Lake City
H. H. Tel: 801.266.2022
W. E. Tel: 800.477.9953

Vermont
H. H. Tel: 800.272.9255

Virginia
H. H. Tel: 800.638.5988

Washington
Seattle
H. H. Tel: 206.882.7000
W. E. Tel: 800.248.9953

Wisconsin
Milwaukee
H. H. Tel: 414.780.7200
W. E. Tel: 800.867.9953

Wyoming
H. H. Tel: 800.332.9326

Sales Offices and Design Resource Centers

LSI Logic Corporation
Corporate Headquarters
Tel: 408.433.8000
Fax: 408.433.8989

NORTH AMERICA

California
Irvine
♦ Tel: 714.553.5600
Fax: 714.474.8101

San Diego
Tel: 619.635.1300
Fax: 619.635.1350

Silicon Valley
Sales Office
Tel: 408.433.8000
Fax: 408.954.3353
Design Center
♦ Tel: 408.433.8000
Fax: 408.433.7695

Colorado
Boulder
Tel: 303.447.3800
Fax: 303.541.0641

Florida
Boca Raton
Tel: 561.989.3236
Fax: 561.989.3237

Illinois
Schaumburg
♦ Tel: 847.995.1600
Fax: 847.995.1622

Kentucky
Bowling Green
Tel: 502.793.0010
Fax: 502.793.0040

Maryland
Bethesda
Tel: 301.897.5800
Fax: 301.897.8389

Massachusetts
Waltham
♦ Tel: 617.890.0180
Fax: 617.890.6158

Minnesota
Minneapolis
♦ Tel: 612.921.8300
Fax: 612.921.8399

New Jersey
Edison
♦ Tel: 908.549.4500
Fax: 908.549.4802

New York
New York
Tel: 716.223.8820
Fax: 716.223.8822

North Carolina
Raleigh
Tel: 919.783.8833
Fax: 919.783.8909

Oregon
Beaverton
Tel: 503.645.0589
Fax: 503.645.6612

Texas
Austin
Tel: 512.388.7294
Fax: 512.388.4171

Dallas
♦ Tel: 972.788.2966
Fax: 972.233.9234

Houston
Tel: 281.379.7800
Fax: 281.379.7818

Washington
Issaquah
Tel: 425.837.1733
Fax: 425.837.1734

Canada
Ontario
Ottawa
♦ Tel: 613.592.1263
Fax: 613.592.3253

Toronto
♦ Tel: 416.620.7400
Fax: 416.620.5005

Quebec
Montreal
♦ Tel: 514.694.2417
Fax: 514.694.2699

INTERNATIONAL

Australia
Reptechnic Pty Ltd
New South Wales
♦ Tel: 612.9953.9844
Fax: 612.9953.9683

Denmark
LSI Logic Development Centre
Ballerup
Tel: 45.44.86.55.55
Fax: 45.44.86.55.56

France
LSI Logic S.A.
Paris
♦ Tel: 33.1.34.63.13.13
Fax: 33.1.34.63.13.19

Germany
LSI Logic GmbH
Munich
♦ Tel: 49.89.4.58.33.0
Fax: 49.89.4.58.33.108

Stuttgart
Tel: 49.711.13.96.90
Fax: 49.711.86.61.428

Hong Kong
AVT Industrial Ltd
Hong Kong
Tel: 852.2428.0008
Fax: 852.2401.2105

India
LogiCAD India Private Ltd
Bangalore
Tel: 91.80.526.2500
Fax: 91.80.338.6591

Israel
LSI Logic
Ramat Hasharon
♦ Tel: 972.3.5.403741
Fax: 972.3.5.403747

Netanya
♦ Tel: 972.9.657190
Fax: 972.9.657194

Italy
LSI Logic S.P.A.
Milano
♦ Tel: 39.39.687371
Fax: 39.39.6057867

Japan
LSI Logic K.K.
Tokyo
♦ Tel: 81.3.5463.7821
Fax: 81.3.5463.7820

Osaka
♦ Tel: 81.6.947.5281
Fax: 81.6.947.5287

Korea
LSI Logic Korea Inc.
Seoul
♦ Tel: 82.2.528.3400
Fax: 82.2.528.2250

Singapore
LSI Logic Pte Ltd
Singapore
♦ Tel: 65.334.9061
Fax: 65.334.4749

Spain
LSI Logic S.A.
Madrid
♦ Tel: 34.1.556.07.09
Fax: 34.1.556.75.65

Sweden
LSI Logic AB
Stockholm
♦ Tel: 46.8.444.15.00
Fax: 46.8.750.66.47

Switzerland
LSI Logic Sulzer AG
Brugg/Biel
Tel: 41.32.536363
Fax: 41.32.536367

Taiwan
LSI Logic Asia-Pacific
Taipei
♦ Tel: 886.2.718.7828
Fax: 886.2.718.8869

Cheng Fong Technology Corporation
Tel: 886.2.910.1180
Fax: 886.2.910.1175

Lumax International Corporation, Ltd
Tel: 886.2.788.3656
Fax: 886.2.788.3568

Macro-Vision Technology Inc.
Tel: 886.2.698.3350
Fax: 886.2.698.3348

United Kingdom
LSI Logic Europe Ltd
Bracknell
♦ Tel: 44.1344.426544
Fax: 44.1344.481039

♦ Sales Offices with Design Resource Centers